LOAN DOCUMENT

PHOTOGRAPH THIS SHEET

AD-A276 967

DTIC ACCESSION NUMBER

LEVEL

1

INVENTORY

ASC-TR-94-5007

DOCUMENT IDENTIFICATION

Aug 93

Approved for public release
Distribution Unlimited

DISTRIBUTION STATEMENT

NTIS GRA&I
DTIC TRAC
UNANNOUNCED
JUSTIFICATION

BY

DISTRIBUTION/

AVAILABILITY CODES

DISTRIBUTION	AVAILABILITY AND/OR SPECIAL
A-1	

DISTRIBUTION STAMP

DTIC
ELECTE
MAR 10 1994
S C D

DATE ACCESSIONED

DATE RETURNED

94 3 9 114

DATE RECEIVED IN DTIC

94-07859

REGISTERED OR CERTIFIED NUMBER

PHOTOGRAPH THIS SHEET AND RETURN TO DTIC-FDAC

DTIC FORM 70A

DOCUMENT PROCESSING SHEET

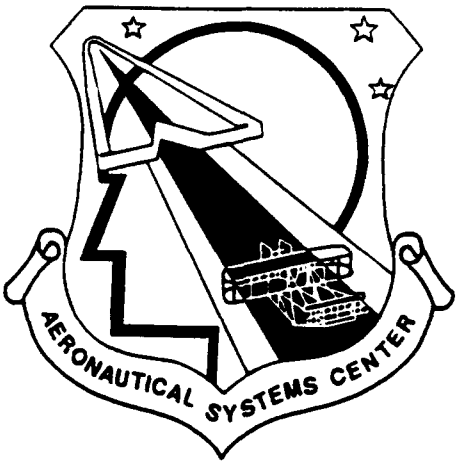
PREVIOUS EDITIONS MAY BE USED UNTIL STOCK IS EXHAUSTED.

LOAN DOCUMENT

HANDLE WITH CARE

ASC-TR-94-5007

MODULAR SIMULATOR SYSTEM (MSS) ENGINEERING DESIGN GUIDE



K KELLY, J BROWN,
G KAMSICKAS, W TUCKER

BOEING DEFENSE AND SPACE GROUP
SIMULATION AND TRAINING SYSTEMS
499 BOEING BLVD
HUNTSVILLE, AL 35824

AUGUST 1993

FINAL REPORT

SYSTEMS ENGINEERING DIVISION
AERONAUTICAL SYSTEMS CENTER
AIR FORCE MATERIEL COMMAND
WRIGHT PATTERSON AFB OH 45433-7126

NOTICE

When Government drawings, specifications, or other data are used for any purpose other than in connection with a definitely Government-related procurement, the United States Government incurs no responsibility or any obligation whatsoever. The fact that the government may have formulated or in any way supplied the said drawings, specifications, or other data, is not to be regarded by implication, or otherwise in any manner construed, as licensing the holder, or any other person or corporation; or as conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

This report is releasable to the National Technical Information Service (NTIS). At NTIS, it will be available to the general public, including foreign nations.

This technical report has been reviewed and is approved for publication.

JEFFREY C. VALITON, Maj, USAF
Program Manager
Special Programs Divsion

JAMES D. BASINGER
Team Leader
Special Programs Division

JAMES J. O'CONNELL
Chief, Systems Engineering Division
Training Systems Program Office

If your address has changed, if you wish to be removed from our mailing list, or if the addressee is no longer employed by your organization please notify ASC/YTSD WPAFB, OH 45433-7111 to help us maintain a current mailing list.

Copies of this report should not be returned unless return is required by security considerations, contractual obligations, or notice on a specific document.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	20 Aug 93	FINAL

4. TITLE AND SUBTITLE

Modular Simulator System (MSS)
Engineering Design Guide

5. FUNDING NUMBERS

F33657-86-C-0149
64227F

6. AUTHOR(S)

K. Kelly, J. Brown
G. Kamsickas, W. Tucker

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Boeing Defense and Space Group
Simulation and Training Systems
499 Boeing Blvd
Huntsville, AL 35824

8. PERFORMING ORGANIZATION REPORT NUMBER

S495-10440-1

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Aeronautical Systems Center
Systems Engineering Division
Bldg 11 2240 B St Ste 7
Wright-Patterson AFB, OH 45433-7111

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

ASC-TR-94-5007

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release;
distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This document is intended as an engineering guide for designing a modular simulator. This guide addresses the definition, history and characteristics of a modular simulator. Systems and Software Engineering activities are defined, and modular simulator segment specific and networking design considerations are discussed. This document discusses the modular simulator program design goals and rationale, design rules and guidelines, and lessons learned that have evolved through the program's demonstration phase. The effective use of this guide, in conjunction with the modular simulator architecture documents referenced herein can substantially reduce the cost, schedule, and risk involved in developing a modular simulator.

14. SUBJECT TERMS

Modular Simulator System (MSS)

15. NUMBER OF PAGES

85

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. Z39-18
298-102

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

This document is intended as an engineering guide for designing a modular simulator (Mod Sim). It is provided in response to contract number F33657-86-C-0149, CDRL 100T. This guide addresses the definition, history and characteristics of a modular simulator. Systems and Software Engineering activities are defined, and modular simulator segment specific and networking design considerations are discussed. This document discusses the Mod Sim program design goals and rationale, design rules and guidelines, and lessons learned that have evolved through the program's demonstration phase. The effective use of this guide, in conjunction with the modular simulator architecture documents referenced herein can substantially reduce the cost, schedule, and risk involved in developing a modular simulator.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

1.1 Purpose. This document is an engineering reference guide for the specification, design and development of modular simulator systems (MSSs). The guide should be used in conjunction with the generic Modular Simulator (Mod Sim) System/Segment Specifications (SSS) and the Modular Simulator System Management Guide.

1.2 Scope. This document identifies the design goals, rationale, rules, and guidelines for an MSS. Where applicable, discussion concludes with lessons learned from the Mod Sim development program. The guide is organized into sections addressing program development, system and software engineering responsibilities, and relevant Mod Sim design issues. The focus of this document is the unique considerations involved with the design and development of a Mod Sim. Those considerations which are relevant to simulators without regard to architecture are not discussed in this guide.

The target audience for this guide is the engineering sector. It contains detailed engineering concepts and terminology. The reader is assumed to be knowledgeable in the areas of systems and software engineering practices and principles as applicable to design of aircrew training devices.

2. REFERENCE DOCUMENTS

2.1 Government Documents.

F33657-86-C-0149 Modular Simulator Design Program Statement of Work

2.2 Non-Government Documents.

S495-10400	System/Segment Specification for the Generic Modular Simulator System Volumes I through XIII
S495-10415	System/Segment Specification for the F-16C Modular Simulator System Volumes I through XII
D495-10436-1	Modular Simulator System Hardware Requirements Document
D495-10437-1	Modular Simulator Design Program, Phase III, Part 2 Final Report
D495-10438-1	Follow On Effort Final Report for the Modular Simulator Design Program
D495-10439-1	Modular Simulator System Management Guide
D495-10735-1	Modular Simulator System Interface Design Document
S495-10734-1	Modular Simulator System Interface Requirements Specification

3. MODULAR SIMULATOR CONCEPT

3.1 Mod Sim Program Background. The development of the Mod Sim architecture began in December 1982. The purpose of this program was to develop and demonstrate a modular design for flight simulators, using standard module functions and communication interfaces. The goal of the program was to demonstrate a generic simulator design, that would reduce future simulator development costs and schedules, and improve simulator supportability. This was accomplished by subcontracting the individual modules to specialists, by reusing existing software, and by providing parallel development and test of the individual modules. The program consisted of three phases, including a Request For Information, a Concept Development Study, and a Concept Demonstration/Validation.

During the first two phases, a conceptual modular simulator architecture was developed and implementation planning was provided. This activity provided a clear indication that the optimum Mod Sim architecture consisted of a distributed processing environment of hardware and software modules connected to non-proprietary bus with standardized interfaces. Boeing proposed this concept to the United States Air Force (USAF), and was selected to continue development in Phase three.

The third phase consisted of two parts, Design and Demonstration/Validation. Boeing, Scientific Applications International Company (SAIC), Rediffusion Simulation Limited (RSL), AAI and Intermetrics made up the design team, that created a generic set of functional definitions. These definitions were based on past simulator interface information, existing training device specifications, Technical Orders (TOs)/Flight Manuals, Federal Aviation Administration (FAA) standards, and contractor experience. A functional dictionary was built , which consisted of the definition and output data for each function.

The Boeing Automated Module Interface Compiler (AMIC) tool was used to determine the optimal allocation of functions to modules, and functional interfaces. As a function was allocated to a module, the output data associated with the function formed the definition of that module's functionality. An initial set of module specifications, consisting of a system level volume plus one volume for each individual module, was generated based on the module functional definitions.

A system performance model was constructed to evaluate alternative hardware communication architectures, and a trade study was performed to determine the architecture that would meet the requirements of real-time simulation.

Throughout the Part 1 design period, Interface Standard Working Group (ISWG) meetings were conducted to evaluate the design effort of the program. These meetings were held at regular intervals in order to involve industry in the development process. The meetings were contractually required and involved approximately 150 companies from the Flight Simulation industry. ISWG participants reviewed the evolving design, offered constructive criticism in the form of action items. The ISWG brought an Industry wide perspective to the Mod Sim program. The resulting architecture represents as much of an Industry consensus as possible, given the diversity and breadth of ISWG

participation. The Statement of Work designated the F-16C as the candidate aircraft for demonstration of the Mod Sim architecture.

After the system level design was completed in Part 1, the Part 2 demonstration period commenced with Boeing subcontracting eight of the eleven segments to AAI, SAIC, and RSL to meet the requirement of 50%-75% sub-contracted effort. The goal of this period was to demonstrate that the individual modules could be independently developed and tested and subsequently integrated at the system level. For this effort, each module was required to reside in a separate hardware chassis as a worst case implementation, aimed at demonstrating the viability of the architecture. An individual VME chassis was chosen for each of the 11 modules. These racks were populated with Motorola single board computers and connected through Fiber Distributed Data Interface (FDDI). As a part of the demonstration effort, Boeing Simulation & Training Systems (S&TS) developed a stand-alone module test device, which was used to test each of the modules prior to system level integration.

The F-16C Mod Sim was demonstrated and subsequently delivered to Williams AFB in December 1990. Since then Boeing has accomplished follow-on work to further the concepts developed during the program. An interoperability study was conducted to provide the capability of connecting modular simulators to a Distributed Interactive Simulation (DIS) environment for team training. This study resulted in the creation of a twelfth segment, "Environment." In addition, a tailoring guide has been developed to aid in the application of the generic specifications to specific programs.

3.2 Modular Simulator Characteristics. A modular simulator has three types of characteristics. These include characteristics that are fundamental to the architecture and do not change across programs, some variable characteristics which afford flexibility in design and implementation, and some general hardware characteristics which are also optional. Figure 3.2-1 illustrates the twelve segment Mod Sim architecture.

3.2.1 Fundamental Characteristics. Four fundamental characteristics are central to the architecture of Modular Simulators.

a. Functional Segments. First, the simulator must be divided into functional segments. (On the Mod Sim demonstrator program, these segments were called modules) Segments are groups of functions or objects that closely related with one another internally, and are loosely coupled to functions in other segments. One measure of internal cohesion is data flow. Objects that pass data to one another are obviously more cohesive than objects that do not share data. Another measure of cohesion is execution order dependency. If one function must always be executed before another, it may be necessary to allocate both functions to the same segment with an explicit requirement to the segment executive as to the order of execution. The allocation of functions to segments is defined in a generic System Segment Specification (SSS). In most cases the functional allocation defined in the generic SSS represent an optimum allocation and should be adhered to if at all possible.

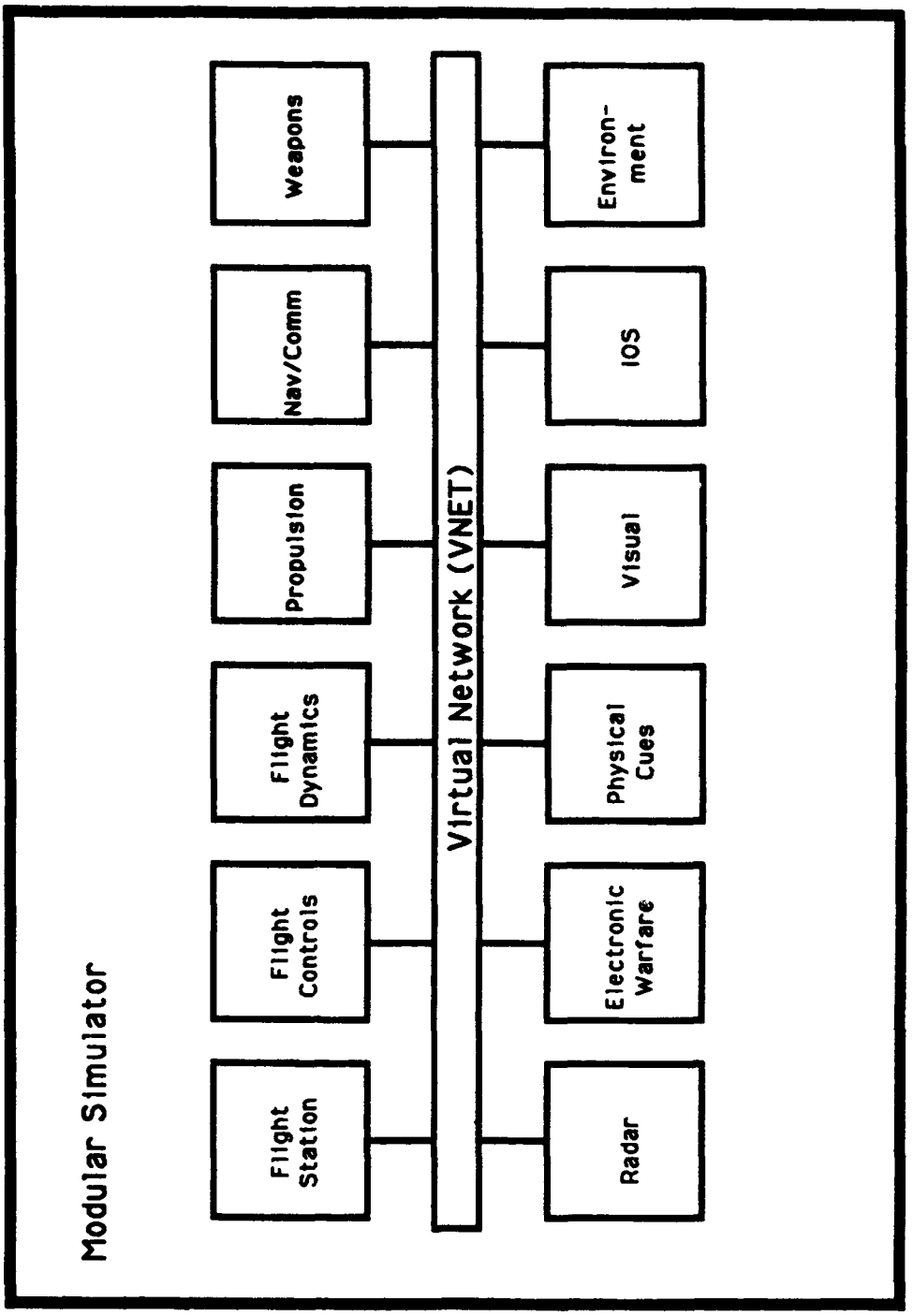


Figure 3.2-1
Generic Mod Sim Twelve Segment Architecture

b. Inter Segment Interfaces. Interfaces in a Mod Sim are unique in that only the output requirements are defined in the Interface Requirements Specification (IRS) along with the required destination(s). The interfaces may be derived from the Mod Sim Interface Design Document (IDD), Appendix A, or may be defined from other systems engineering analyses. Nevertheless, all inter segment messages, back to the root types, must be defined.

Inter segment communication in a Mod Sim is accomplished by means of a message based architecture, not a shared memory architecture. Segment applications may not write into a shared memory area which is readable by other segment applications. Communication within a segment may be implemented using shared memory, but a segment application program sends and receives messages.

The interfaces and message based communication methods described above must be the only means that segments use to communicate with one another. Segments should not communicate with other segments through a "back door" interface unless an overriding requirement exists (back door interfaces are discussed in Section 4). Just as functions have been decomposed and allocated in the SSS, messages have been predefined in the IRS and can generally be used as is. The IRS can be tailored, but the less tailoring, the better. Any such tailoring should consider the implications of future reuse and compatibility with existing reusable segments.

c. Timing. All timing requirements must be strictly specified in Appendix B of the MSS IDD. The general assumption is that a message will be received no later than the frame following transmission. There can be cases where the message is required in the same frame. In this event, the sending segment must transmit no later than a specified number of milliseconds into the frame and the receiving segment cannot execute the dependent function prior to a specified number of milliseconds into the frame.

d. Stand Alone Test. There must be a capability to test each segment as a stand-alone system. A segment tester must be developed to simulate all of the other segments in the system. It must supply the segment under test with all synchronization messages (e.g., a clock tick message) and all mode and state transition messages. It must also supply the segment under test with input data and have the capability of collecting output data from the segment under test. The test functions must be able to operate in real-time in order to test segment requirements to compute responses within time limits. The segment tester must allow the operator to generate input data files, and to display output data of the segment under test.

e. Virtual Network. While the hardware and software implementations of the communication architecture are optional, the requirement for a Virtual Network (VNET) is not. Every segment must appear to be communicating over a network. The VNET is a concept that was developed, implemented and demonstrated on the Mod Sim program and is a requirement for any Mod Sim.

3.2.2 Variable Characteristics. There are several characteristics within a Mod Sim that have some degree of variability. They include the method of decomposition, the number of segments within a given Mod Sim, type and number of computational systems, and the VNET implementation.

a. Segment Decomposition. Decomposition refers to the allocation of functional requirements to segments. For any simulator, decomposition is required. However, the method of decomposition is optional. The Mod Sim program employed a functional decomposition that resulted in twelve modules. Other methods of requirements decomposition and allocation are available, such as objects, systems, subsystems, etc. Regardless of the method used, the final step is to allocate messages to the resulting segments.

b. Number of Segments. The composition and number of segments are not fixed within a Mod Sim architecture. Unless a full fidelity weapon system trainer is being developed, it is probable that one or more segments are not required in most applications. It is possible to remove entire segments and still retain a Mod Sim architecture. For example, a transport aircraft cockpit procedures trainer Mod Sim may have no Visual, Electronic Warfare, or Weapon Systems segments. Similarly, a lunar rover simulator may replace the Flight Dynamics Segment with a Vehicle Dynamics Segment. It is also permissible to combine two or more segments (Flight Controls and Flight Dynamics, for example) into one module.

c. Computational Systems. The number of computational systems or CPUs is not governed by the Mod Sim architecture, since Mod Sim is a software architecture. It is possible to combine any number of, or all segments to run on one system or one processor. The allocation of computational resources only as required by various segments, is one of the strengths of the Mod Sim architecture. In the Mod Sim demonstration program, each segment was implemented as a separate, multiprocessor hardware module in order to test the worst case of one computational system per segment.

d. Virtual Network Implementation. A physical network is not essential to Mod Sim, but a virtual network (VNET) is. Segments may communicate with one another over a back plane, or through common memory, or through any method or combination of methods, so long as each segment application believes it is sending and receiving messages over a network. On the Mod Sim program, FDDI was tested and demonstrated to be a viable network media.

3.2.3 Hardware Characteristics. The Mod Sim architecture is not dependent on the utilization of specific hardware. The implementation of each software segment into a VME based computational system is only one of many possible hardware configurations. The software segments may be combined into one or more VME compatible computational systems, may be driven by a single mainframe processor, or may be installed on another appropriate hardware configuration.

The hardware architecture chosen for each new Mod Sim project or application should be the focus of program trade studies. In the rapidly changing world of computational components, each new program should assess products in light of factors such as availability, supportability, cost, performance, and compatibility with software interfaces.

3.3 Mod Sim Architecture Selection. The decision to select the Mod Sim architecture for a simulator or family of simulators should be made during the conceptualization phase of a program, as software, hardware and engineering reusability may significantly impact the cost of the program. This design decision should determine if a Mod Sim architecture is the most practical, and if it is, the specific implementation of the architecture. It is recommended that software architecture trade studies be conducted to determine the feasibility of utilizing the Mod Sim architecture, and the optimum architecture to implement. Trade study factors should include:

a. Cost. The cost of tailoring Mod Sim systems engineering products (Specifications, SOW, Interface design, etc.) and software engineering products must be weighed against alternative architectures. Intuitively, tailoring should offer considerable savings over any approach involving complete design development.

b. Schedule. The schedule impact of tailoring Mod Sim products must be considered. Not only is time saved in tailoring existing design and specification products, but the parallel development of the individual segments can contribute to schedule compression.

c. Subcontracting requirements. Designers of a program with significant subcontracting requirements must consider how the trainer is subcontracted when designing and testing the devices. The Mod Sim architecture lends itself well to the logistics of a highly subcontracted development program.

d. Program Dynamics. A training system that is likely to experience a large number of updates through the program life cycle also lends itself well to the Mod Sim architecture, due to the stability of well defined interfaces and the loose coupling between segments.

e. Software Reuse. The amount of software reuse between devices in a training system family should be a consideration. For example, modular reuse between a WST and CPT makes the Mod Sim architecture attractive. Reuse or modification of existing software from other programs should also be considered.

3.4 Mod Sim Demonstrator Program (F-16C). The discussion that follows illustrate an application of Mod Sim to a specific aircraft simulator. Though answers may change, the process of trade studies and consideration gives some insight to the issues faced by the application engineer. The Mod Sim demonstration program hardware architecture , shown in Figure 3.4-1, implemented each software segment into a Versa Module European (VME) computational system . This single segment per system architecture was dictated by program requirements as the worst case (interface requirement). The computational systems were selected as the result of program trade studies, and linked through a fiber optic bus, known as the Fiber Distributed Data Interface (FDDI).

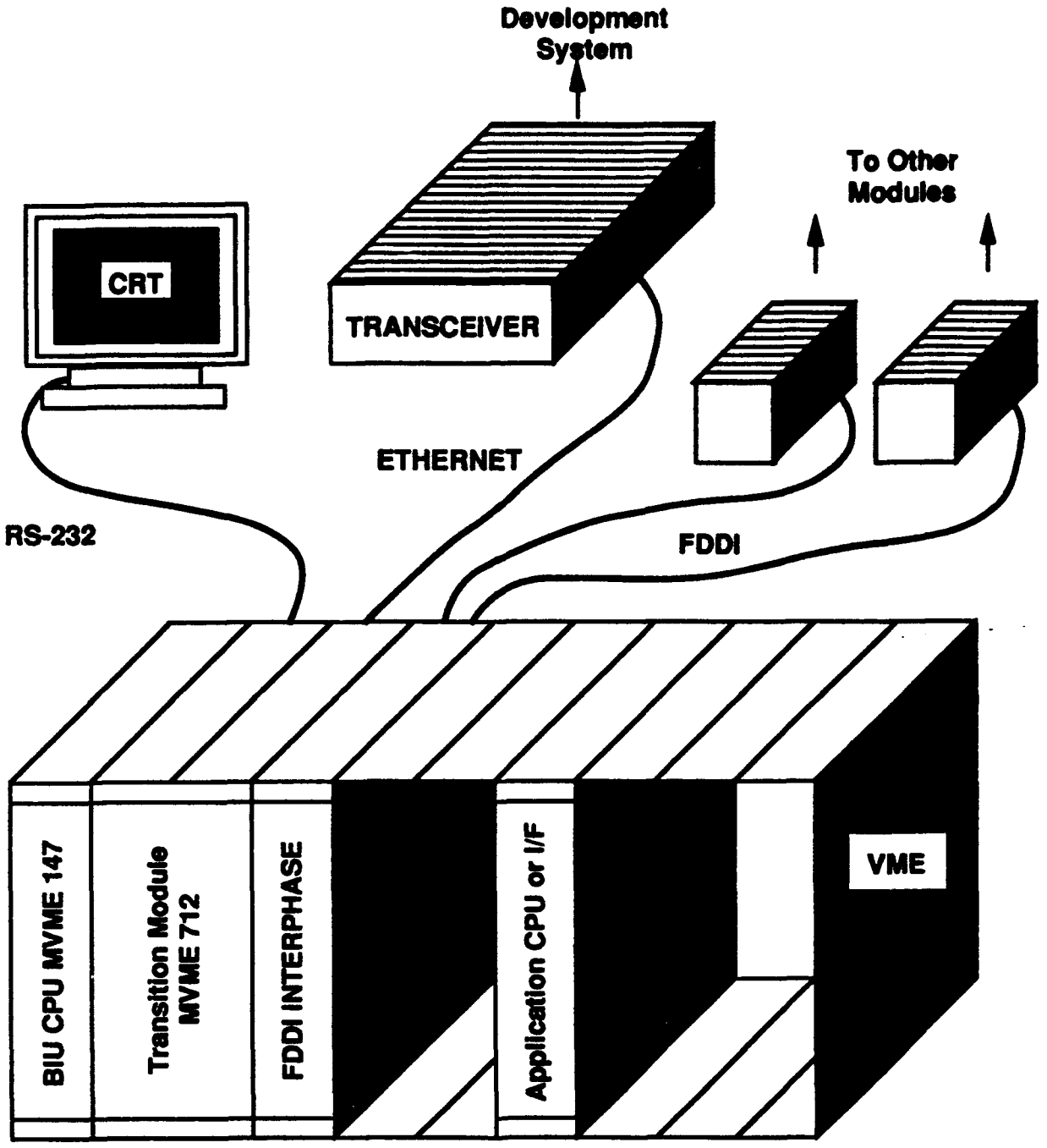


Figure 3.4-1
F-16C Demonstrator Hardware Architecture

3.4.1 F-16C Communication Architecture. The VME bus was chosen as the internal back plane for the Phase III, Part 2, F-16C MSS demonstrator modules to satisfy the need for internal communication between the different processors and memory boards of the individual modules for the following reasons:

a. VME is a standard non-proprietary bus.

b. VME products are sold by over 100 vendors.

c. Cross compilers for the VME products are available from several vendors.

d. Advances in hardware and software appear earlier on VME formats than on other bus architectures.

e. VME back plane speed has a sustained 80 megabits/second rate, which was determined to be more than sufficient for the individual modules.

f. Reasonable cost for required performance.

The processor board chosen for the Bus Interface Unit (BIU) was the MVME147, which contains the Motorola MC68030 microprocessor and the MC68882 math coprocessor. The MC68030 microprocessor was selected over other microprocessors because of its ability to execute instructions faster, interrupt latency was less, it was compatible with mature ADA and C compilers and development tools, and the developer was familiar with the product. The MVME147 Computer Processing Unit (CPU) board also offered the Advanced Micro Devices (AMD) Lance Ethernet chip and interface, up to four megabytes of dual ported Random Access Memory (RAM), RS-232 ports, and supports the VADS/Works operating system.

The VADS/Works operating system (OS) by Verdix and Wind River Systems was chosen over Ready Systems and Motorola products for the Mod Sim F-16C demonstration. This OS was chosen because it was already available and fully functional, it closely resembled Unix, which was important because the Xpress Transfer Protocol (XTP) was delivered in the Unix environment first, and it offered development flexibility.

A trade study of alternative FDDI boards selected the Interphase board due to cost, and satisfaction of technical requirements. Alternative products cost more due to dual ring capabilities, on-board CPUs, and new product development costs. The simpler, cheaper Interphase product adequately satisfied Mod Sim program needs.

3.4.2 F-16C Network Interface Hardware. The network interface developed during the Mod Sim Program was the BIU. The BIU consists of a set of communication protocols and data manipulation routines as shown in Figure 3.4.2-1. It is recommended that the hardware chosen for the application BIU be compatible with the application processors. For example, the BIU for each module in the Mod Sim consisted of a VME chassis and power supply, an Interphase 3211 Falcon FDDI board, a Motorola MVME147 CPU

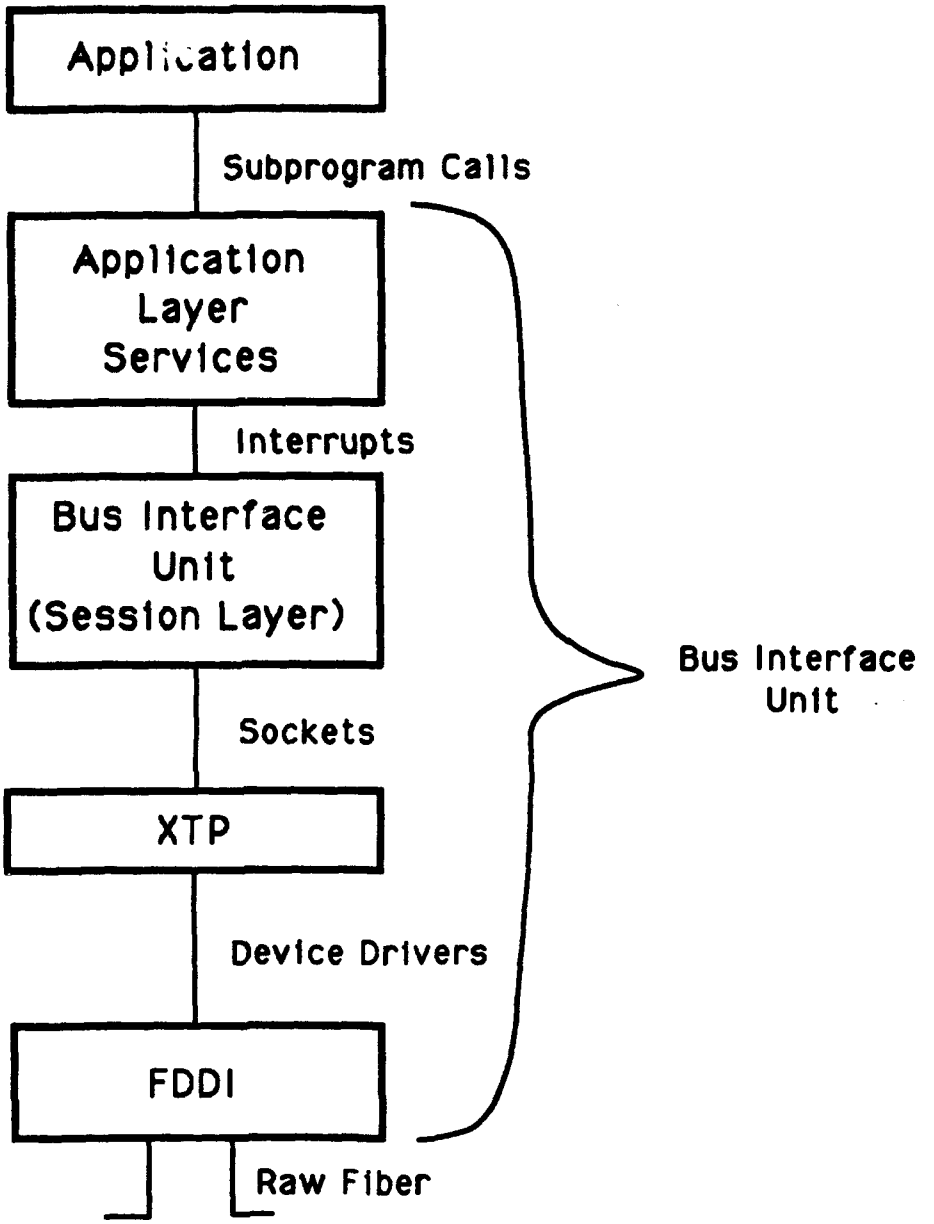


Figure 3.4.2-1
F-16C Mod Sim Internal BIU Layering

board, and an MVME712 transition module. The transition module provided RS232 and Ethernet serial connections to the application software CPU. The Mod Sim BIU hardware allocation is shown in Figure 3.4.2-2. Each of the modules was connected through a fiber optic cable for real-time simulation, and via Ethernet for uploading and downloading of simulation programs while simulation was not being performed.

Trade studies were conducted and a System Performance Model (SPM) was developed during Phase III, Part 1, that determined the maximum allowable latency for the BIU is 550 microseconds. BIU latency is measured as the time that a BIU can take to receive a message for transmission from the application , pack the message and pass it to the FDDI, or to receive a packed message from the FDDI, place it in memory, and notify the application that it is present.

On the Mod Sim device, which implemented or simulated message traffic for each of the twelve modules individually, FDDI proved to be a more than adequate network media. The speed of transmission met all requirements, and the timing of the transmission was deterministic, unlike Ethernet, which is a collision detection protocol.

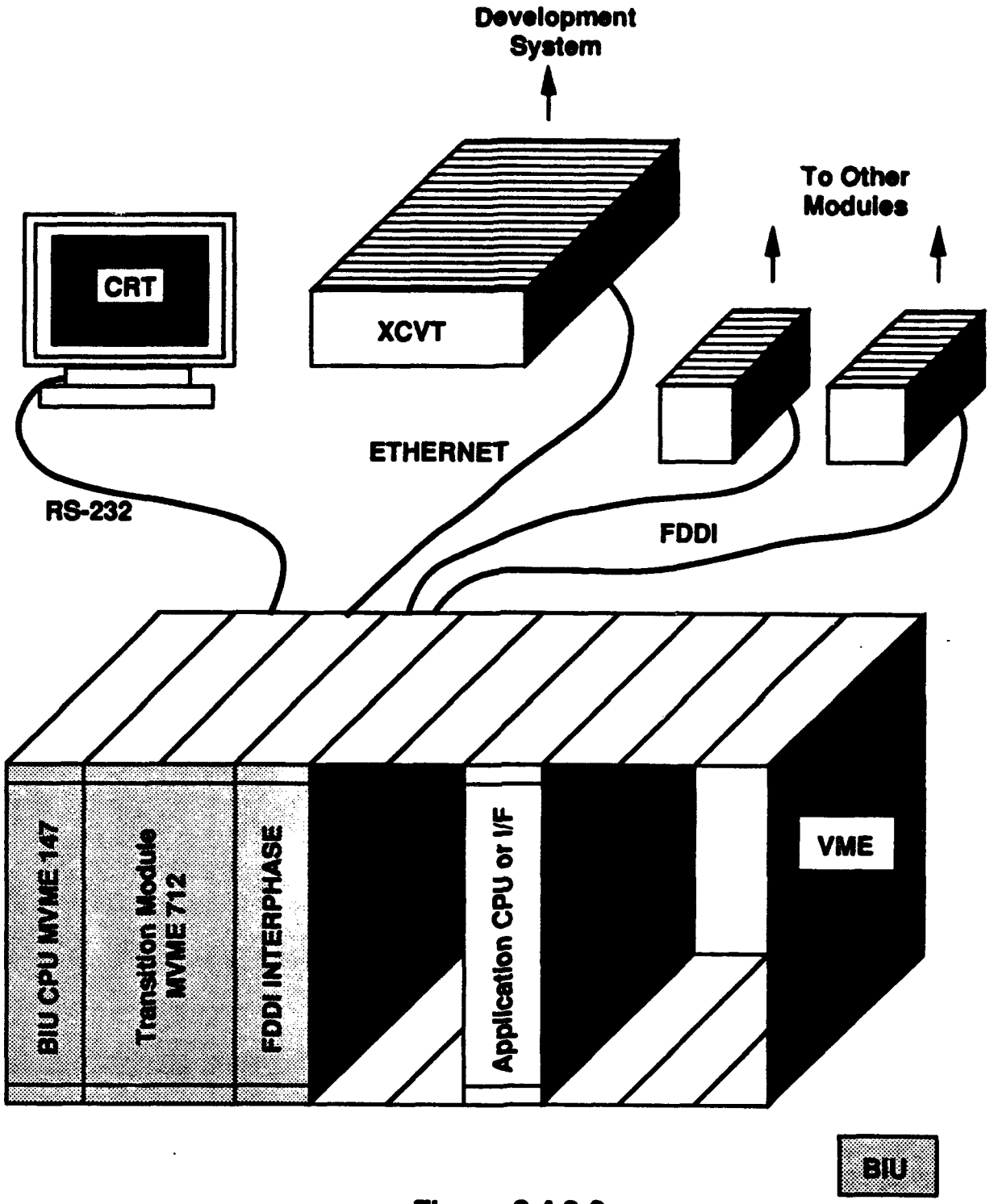


Figure 3.4.2-2
BIU Hardware Allocation

4. SYSTEMS ENGINEERING ACTIVITIES

Systems Engineering activities consist of analyzing functional requirements and translating them into performance requirements, defining system level interfaces, and defining hardware and software requirements. Systems Engineering must analyze the simulation system requirements, and perform trade studies to determine the optimal software system architecture and hardware configuration to meet those requirements. For the purpose of this section, it is assumed that the systems engineering process, that yielded a Mod Sim software architecture as the solution, has been completed.

In accordance with MIL-STD-490, Systems Engineering is responsible for preparing and submitting a System (Type A) Specification and Development (Type B) Specifications. For a modular simulator program, this task consists of tailoring the Mod Sim Generic System/Segment Specification, S495-10400, Volumes 1-13. Although each of the volumes in the generic specification are formatted as Type A specifications in accordance with DI CMAN-80008A, they provide excellent starting points for Type B development specifications.

4.1 Requirements Analysis. Systems Engineering defines a base set of functional requirements, often provided in a System Specification, defines associated system performance requirements, and ultimately derives design requirements. When a Mod Sim approach is selected, a superset of performance and design requirements already exists in the generic SSS with tailoring instructions embedded therein. Systems Engineering documents the requirements analysis by tailoring the SSS to the application specific needs. This tailoring involves deletion of non-required functions, possible modification of existing functions and the addition of potential new functions to meet the specific needs of the aircraft being simulated. The existence of these predefined requirements significantly reduces the systems engineering activity by reducing the requirements analysis effort. In fact, both the system and segment level requirements can be predominantly defined during the proposal preparation phase rather than at preliminary design review.

Not only are the requirements defined early with respect to traditional simulator programs, but even an earlier definition of interfaces is possible. Like the requirements in the Generic SSS, Appendix A of the IDD contains a superset of interface requirements which merely require tailoring to the specific application. Early definition of interface requirements will contribute in a large measure to the success of subsequent integration activities.

4.1.1 Required Segments. The first step in defining a Mod Sim is to determine which of the traditional twelve segments are required. The required segments are determined by several factors. The most significant factor is the air vehicle system complement and associated capabilities. For instance, a device simulating a cargo aircraft would not necessarily require an Electronic Warfare segment. An equally important factor is the actual training capability required . A device used for training instrument procedures would not necessarily require a Visual segment or a Physical Cues segment. As a

minimum, a Mod Sim must have at least two segments, one of which is a control segment, and the other a simulation segment. This would be a radical tailoring of the specification and would only be applicable to devices such as part task trainers.

4.1.2 Functional Requirements Allocation to Segments. The generic Mod Sim System/Segment Specification (SSS) contains an allocation of functional requirements to the twelve individual segments. This functional allocation was determined through a rigorous analysis of design factors for air vehicle simulators. The basic concepts should apply to any application. The criteria included coupling requirements, bus traffic/interface requirements, vendor specialization, data/design criteria segmentation, and Industry/Government preferences. This allocation is by no means inviolate, but departures from it must be considered judiciously, as interfaces will invariably be affected. There is a class of functions that have optional allocations. An example is height above terrain. This function can be allocated to the Environment, Radar, Navigation or Visual segments. A related trade study, discussed in Section 4.9, is recommended to help determine the service function allocations. Another consideration in functional allocation is the degree of coupling required between certain functions. Loose coupling enhances integration, by minimizing inter segment dependencies, and may enhance reusability of the segment.

4.1.3 Module/Segment Configuration Trade Study. A specific trade study required in tailoring a Mod Sim application concerns the appropriate combining of segments into modules. Segments may be combined for several reasons including contract work share, media boundaries, latency requirements, hardware and associated costs, and external interfaces, to name a few. For example, emerging visual systems often accommodate front end processors which host air vehicle flight simulation. Special care must be exercised when combining segments as reusability can be affected.

Figure 4.1.3-1 is a sample Mod Sim in which the twelve segments were allocated between three modules. The Simulator System module contains ten of the segments (Figure 4.1.3-2), while the Flight Station and Visual System modules contain one each. The factors that would most likely result in this configuration are contract work share and the need to minimize hardware processor cost. It should be noted that the Simulator System module could easily be separated by reallocating segments should future upgrades require additional processing capacity.

4.1.4 Specification Tree. As soon as the module/segment configuration is defined a specification tree should be developed. The tree provides a top level configuration of the system, recognizable development boundaries and their associated interfaces, and a road map for requirements development and completion. An example of a specification tree for a program with a family of trainers is illustrated in Figure 4.1.4-1. In this example, the propulsion, flight dynamics and navigation segments of two different devices could be developed from identical specifications and therefore be reusable between the two devices, while the visual segment specifications could be unique to each device and would require separate specifications.

The specification tree in Figure 4.1.4-1 assigns a Type A System Specification at the Training System level, whereas the generic SSS assumes a single training device type is the System. Consequently, the generic SSS may have to be dissected and the

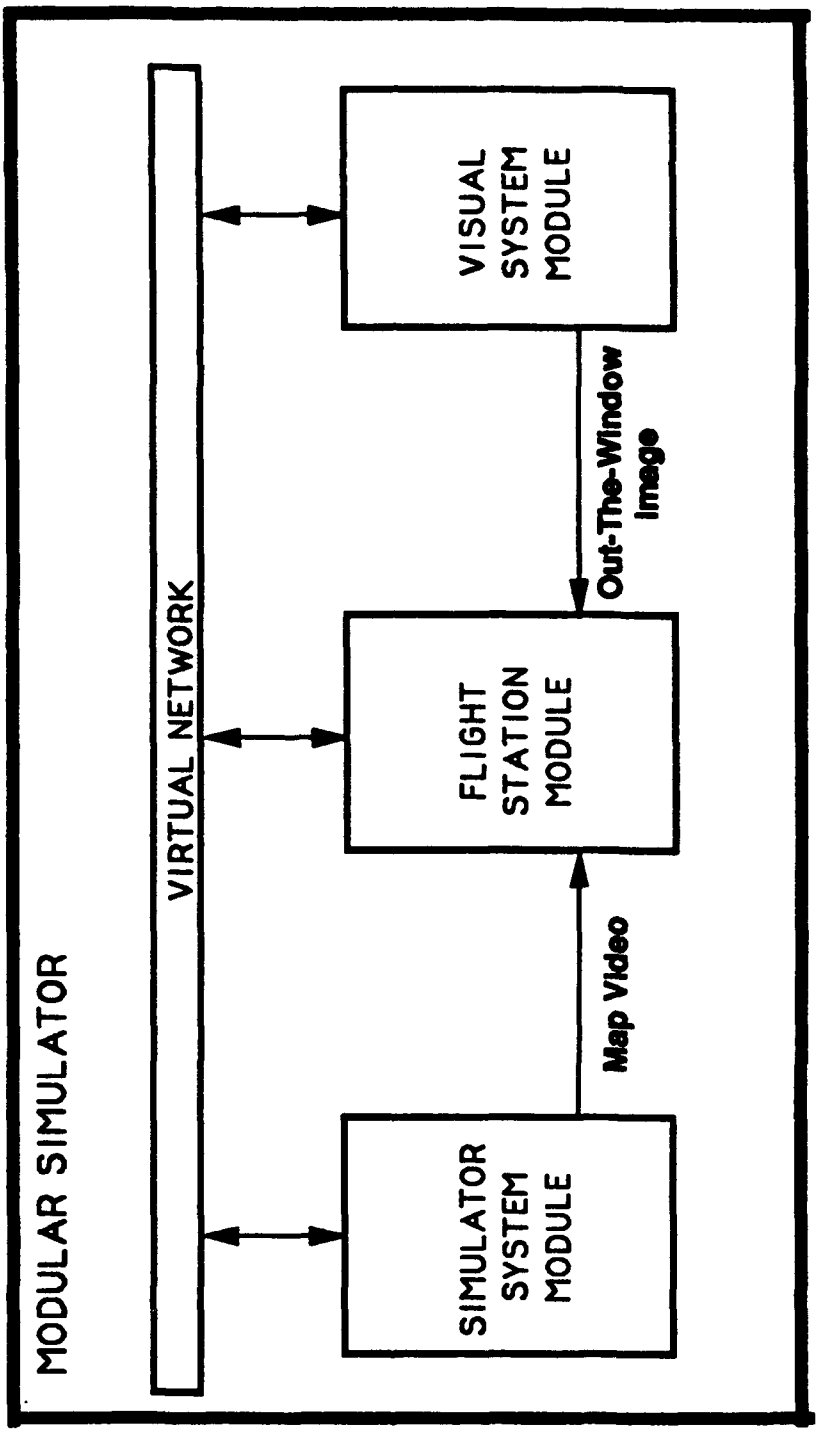


FIGURE 4.1.3-1
Sample Mod Sim Module Architecture

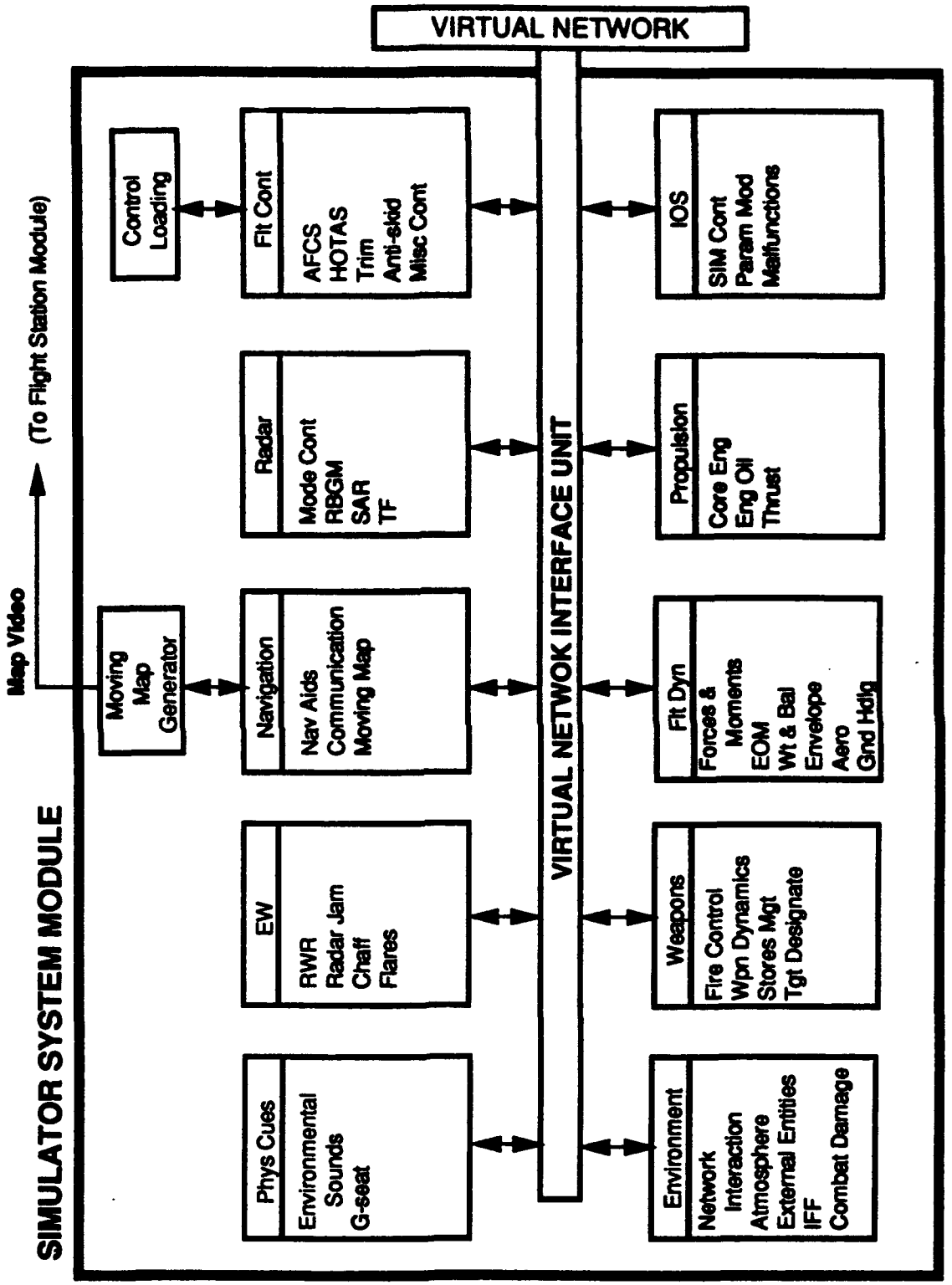


FIGURE 4.1.3-2
Sample Module Configuration

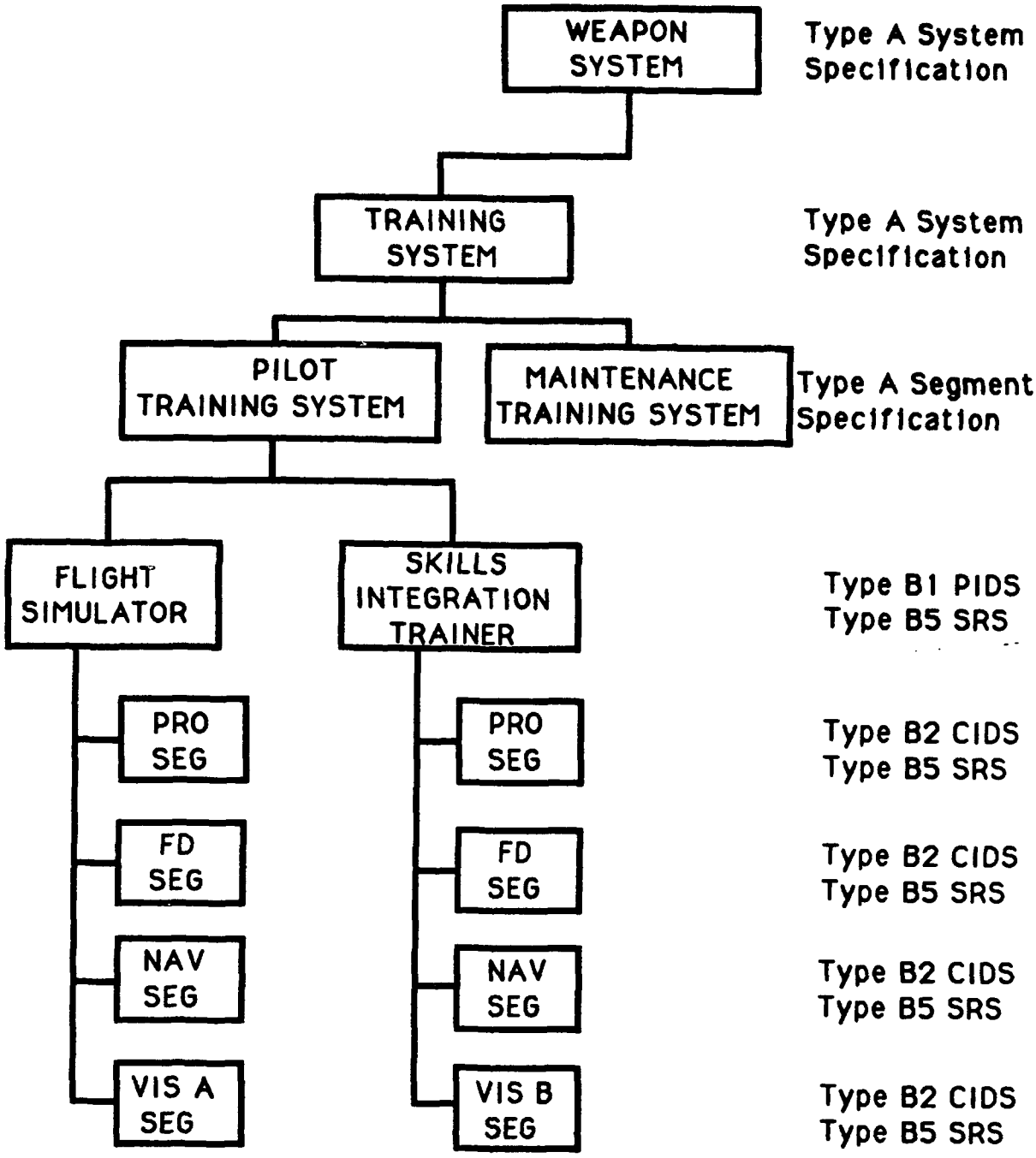


FIGURE 4.1.4-1
Sample Specification Tree

requirements apportioned to the various system levels as appropriate. If a common architecture is imposed at the Training System level, then Volume I architecture requirements would have to allocated to the System Specification. If for some reason different architectures are selected for the Pilot Training System and Maintenance Training System, then the SSS Volume I requirements would have to be uniquely applied to the two Segment Specifications. In any case, Volumes II through XIII will be allocated to the individual segments probably in the form of Critical Item Development Specifications (CIDS) and Software Requirements Specifications (SRSs) as illustrated in the tree.

4.1.5 System Software Architecture. The Mod Sim architecture is a functional allocation of up to thirteen (12 segments and a virtual network controller) Computer Software Configuration Items (CSCIs). The segment CSCIs communicate with each other by means of messages on a virtual network. Further, software architecture internal to each of the segments is left to the segment developer. Section 4.4 discusses alternative segment software architectures, and Section 5.1.1 discusses the experience gained on the Mod Sim demonstrator program with them.

4.1.6 Timing Requirements. Timing requirements are primarily driven by cue correlation requirements. These requirements are critical factors in determining the final allocation between segments and modules and the fundamental frame timing (iteration rate). Cue correlation requirements may vary from simple control input to instrument response on a part task trainer to more complex multiple interrelated events on a full fidelity weapon system trainer (WST). An example is the B-1 WST, whose correlation requirement involved visual transport delay, motion transport delay, and instrument response. Requirements existed at the subsystem level for each of these items and at the system level between the subsystems. Specifically, the visual system requirement was a transport delay of no more than 117 milliseconds, a motion system delay of no more than 50 milliseconds, with a restriction that the visual response in no case could precede the motion response. With the fundamental MSS assumption that a message transmitted in one frame must be received by the next frame, the message critical paths must be defined and coupling decisions made accordingly. This also requires that a frame rate be defined which may subsequently be changed during the frame timing trade study discussed in Section 4.4.2.

In addition to cue correlation requirements, there are subsystem operational requirements that relate to latency. For instance, a simulated laser designator function would normally require a service from an intervisibility server to determine which object in a sensor data base is being "lased". If the service request takes too long getting to the server, and the object is a high velocity moving model, lock-on may not occur since the object may have moved out of the laser LOS cone defined in the request. This example might dictate that the laser ranging service of the intervisibility function be allocated to the Radar segment to be collocated with the laser designator function. The results of the latency analyses represent a significant input to the frame timing trade study referenced above.

4.1.7 Selective Fidelity Requirements. Selective fidelity refers to varying degrees of fidelity for the individual simulation models that make up a simulator. Selective fidelity

should be considered where families of trainers are involved, such as weapon system trainers, mission trainers, part task trainers, desk top trainers, and others. Typically a WST function would have the higher fidelity requirements and consequently are more complex, requiring greater computational resources. Trades are recommended to determine whether the additional computational resources, required for the higher fidelity software, offset the cost of unique software for a lower fidelity device, such as a CPT, with lesser computational resources. Experience has shown that reuse of high fidelity software in lower fidelity devices is generally preferable to maintaining multiple software baselines with its associated life-cycle costs.

4.1.8 Multi-simulator Networks. Many emerging simulators are intended to support multi-participant or interoperable exercises over a network such as Simulation Network (SIMNET) or Distributed Interactive Simulation (DIS). The Mod Sim architecture includes a twelfth segment called the Environment segment, which accommodates such network interface functions. The Environment segment should also perform the tactical scenario and the platform and signal environment (hostile and friendly) functions along with some of the natural environment functions in both the networked (multi-participant) and autonomous (single participant) training modes.

The fundamental difference in functionality of this segment when operating in the network mode versus the autonomous modes, is that in the network mode the simulated external environment is derived from network environment message traffic, while in the autonomous mode the segment must actually simulate the external environment.

4.1.9 Sensor Data Base Allocation. In a Mod Sim, control of the visual and radar data bases, threat environment, and navigation database is normally performed by the Database Management functions within the Environment segment. These functions contain all pertinent information regarding the access, modification, and level of detail of the databases to be used in the simulation. When allocating sensor and visual data bases to the Environment segment, it is important that the interface for data base transmission to the sensor simulation systems or image generator not introduce any additional latency than if the data bases were integral to those systems, as is normally the case in a non-Mod Sim program. This is accomplished by the use of back door interfaces between the data bases and the various systems. Back door interfacing of the data bases is also required to prevent VNET saturation with data for image generation systems.

Another reason for allocating the various data bases to this segment is that, in the future, operation on SIMNET or a DIS based network will probably require the sensor data bases to be dynamic with real-time changes defined from the network.

4.2 System Level Interface Definition. Generic system level interfaces are defined in Appendix A of the IDD. They include the Mod Sim global types (interface data common to all segments) and the individual segment interfaces. It is intended that Appendix A be tailored to the application specific requirements using the embedded tailoring instructions provided in the IDD.

4.2.1 Virtual Network Requirements. The Mod Sim architecture requires all inter-segment data to be passed in the form of messages, as defined in Appendix A of the

IDD, on a Virtual Network (VNET). The VNET may be implemented by a physical network, such as FDDI or Ethernet, shared memory, by a hardware back plane, or any other communication architecture. The primary goal is to make the implementation of the VNET transparent to the individual segments. The Mod Sim architecture permits multiple segments to reside on the same computational system. When this occurs it is desirable for the collocated segments to communicate with each other using the same services as segments residing on the other processors within the Mod Sim complex. A segment should not be able to differentiate the processor it is residing on, or whether or not other segments are hosted on the same processor. In general, the segment should be hardware independent.

4.2.2 Multi-segment Module Interface. When analyses indicate that segments should be combined into modules an additional interface must be defined between the Mod Sim VNET and the individual segments and it must be transparent to the segments involved. A module interface receives incoming messages from the Mod Sim VNET and directs them to the appropriate segment(s) within the module. For outgoing message from the constituent segments the interface will determine which messages need to be broadcast to the VNET, retained within the module or both.

Figure 4.2.2-1 illustrates the communication of a weapon reset message from the IOS to the Weapon segment. The IOS Application first requests the Application Services to send the message, then Application Services builds the message and requests the VNET interface to transmit the message to the network, and then the module VNET interface determines that the message need only be transmitted back to the module. The interface then writes the message into the Weapon segment message memory and notifies Weapon Application Services that the message is available. The bold lines indicates the apparent path of data flow, while the dashed line indicates the actual path.

Figure 4.2.2-2 illustrates the communication of a runway light message from the IOS to the Visual segment. Again the IOS Application first requests the Application Services to send the message, then Application Services builds the message and requests the VNET interface to transmit the message to the network, and then the module VNET interface determines that the message need only be transmitted to the VNET. In this instance the apparent and actual communication are one and the same.

Figure 4.2.2-3 illustrates the communication of an aircraft position message from the Flight Dynamics to the Visual and IOS segments. In this example the VNET interface determines that the message needs to be retransmitted to the sending module and to the VNET. The interface then writes the message into the IOS segment message memory and notifies IOS Application Services that the message is available and sends it to the VNET.

4.2.3 Segment to Segment Interface. Segment interfaces, defined in Appendix A of the IDD, should apply to most simulators for fixed, variable or rotary wing aircraft. However, some interface tailoring is anticipated to meet specific application requirements. Every effort should be made to use the reusable interface provided Appendix A of the IDD to define the aircraft application. Addition of new messages and data structures for specific applications may reduce the reusability of the segments. However, if a new

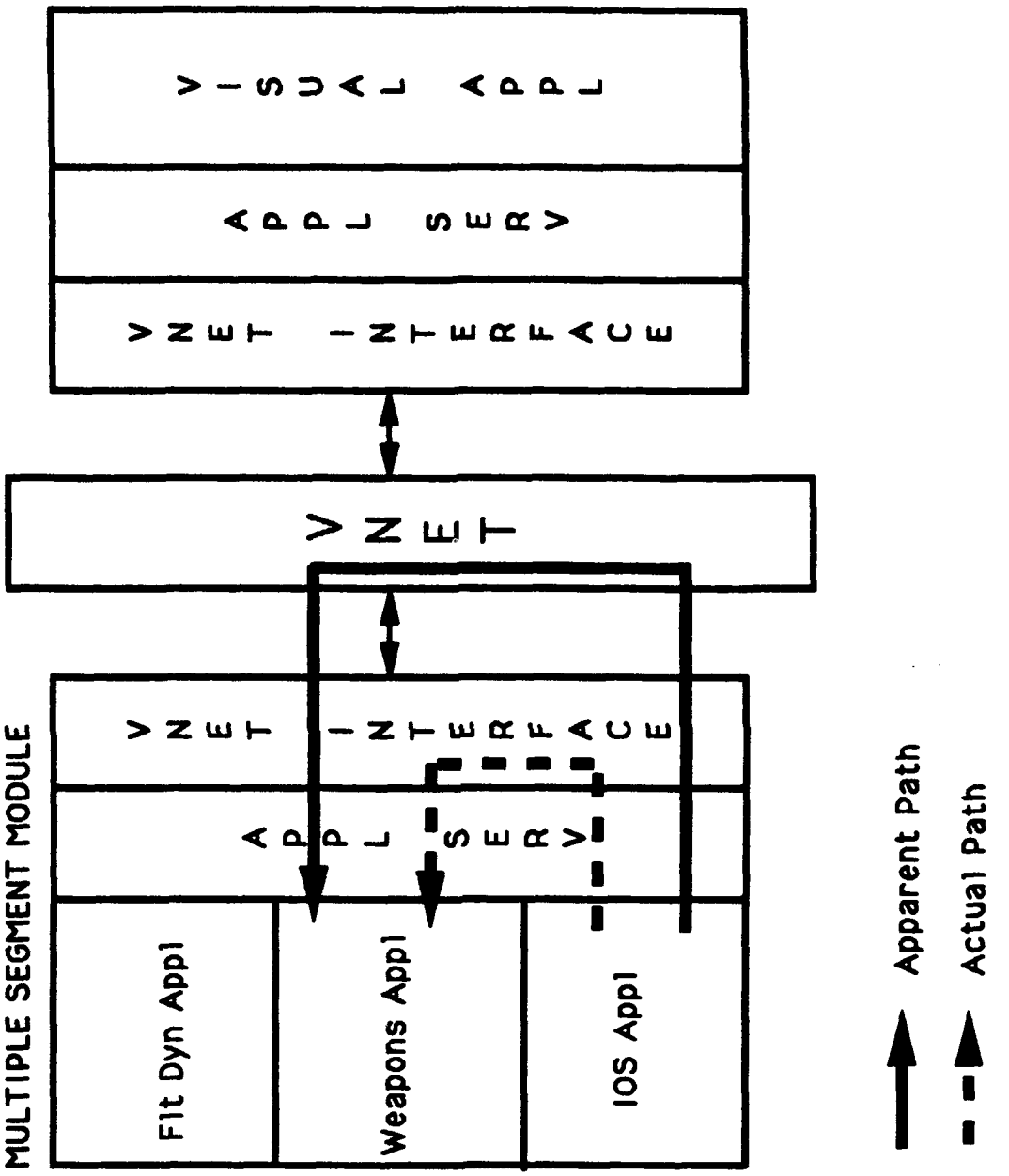


Figure 4.2.2-1
Multi-Segment Module Communication
Weapon Reset Message Example

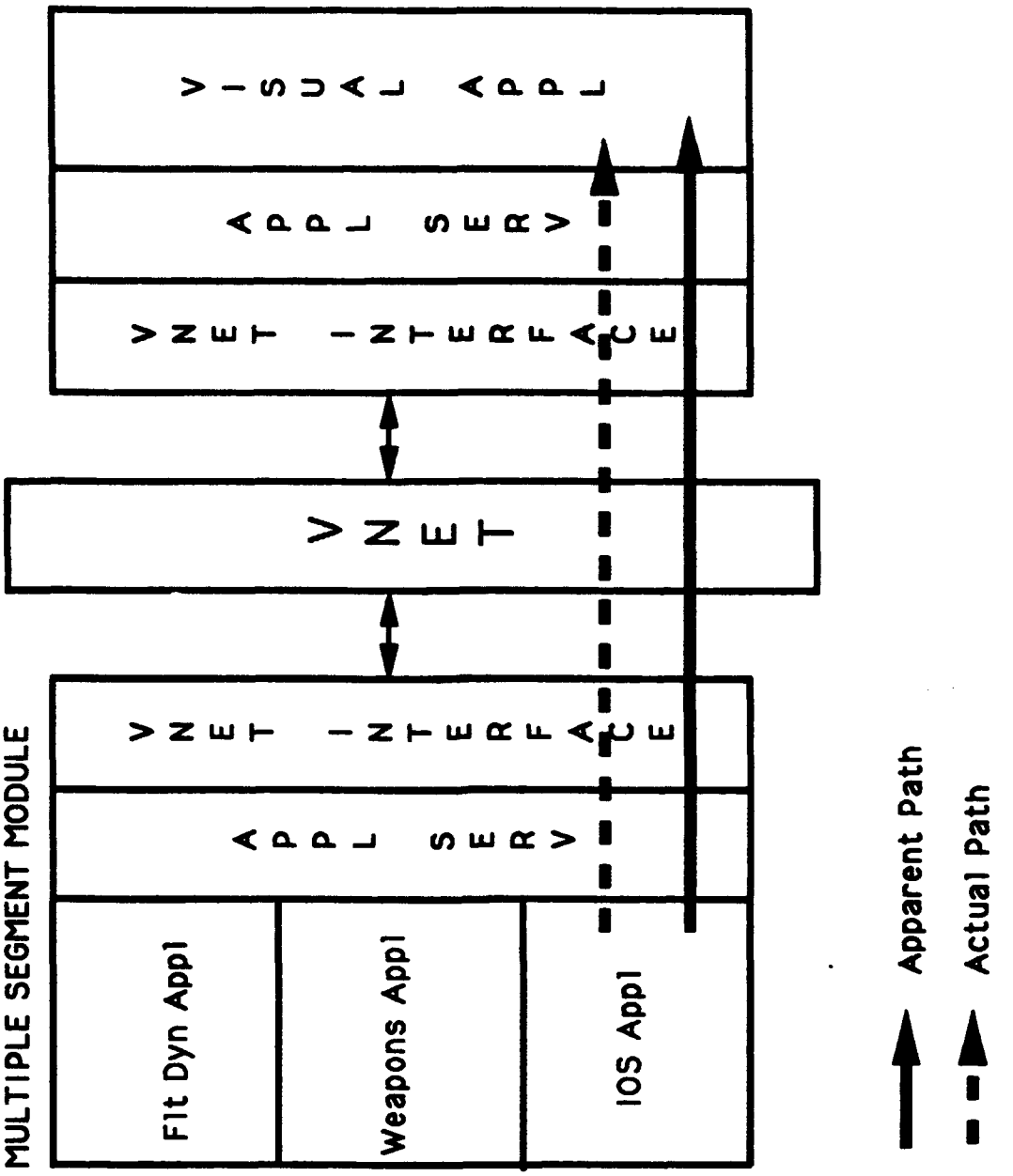


Figure 4.2.2-2
Multi-Segment Module Communication
Runway Lighting Message Example

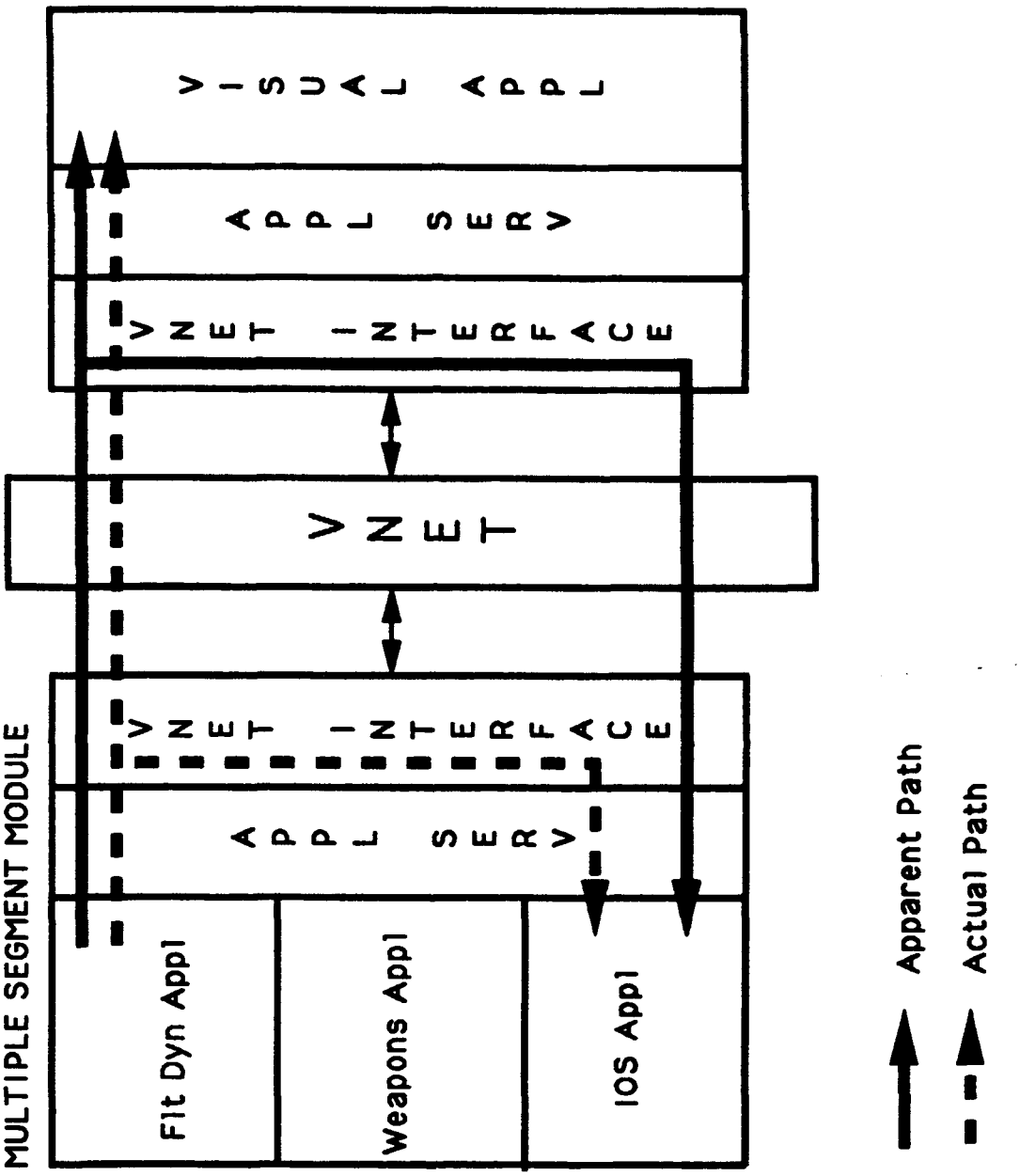


Figure 4.2.2-3
Multi-Segment Module Communication
Aircraft Position Message Example

message is required, it should be defined as generic as possible to promote future reuse.

4.2.4 Back-Door Interfaces. A back-door interface is any interface between segments or modules which does not communicate over the VNET. Back-door interfaces are required when the required data rate exceeds the VNET bandwidth or when the data format is incompatible with network transmission medium. An obvious example is pixel data or RS-170A video from the image generator in the Visual segment to a display in the Flight Station segment. Another example is a 1553 bus between stimulated avionics processors. In this instance, the prime consideration is the coupling required between the processors as avionics communication rates may be much higher than the device frame rate. Fly-by-wire control systems typically operate at 500 Hz, which is significantly higher than any normal device frame rates.

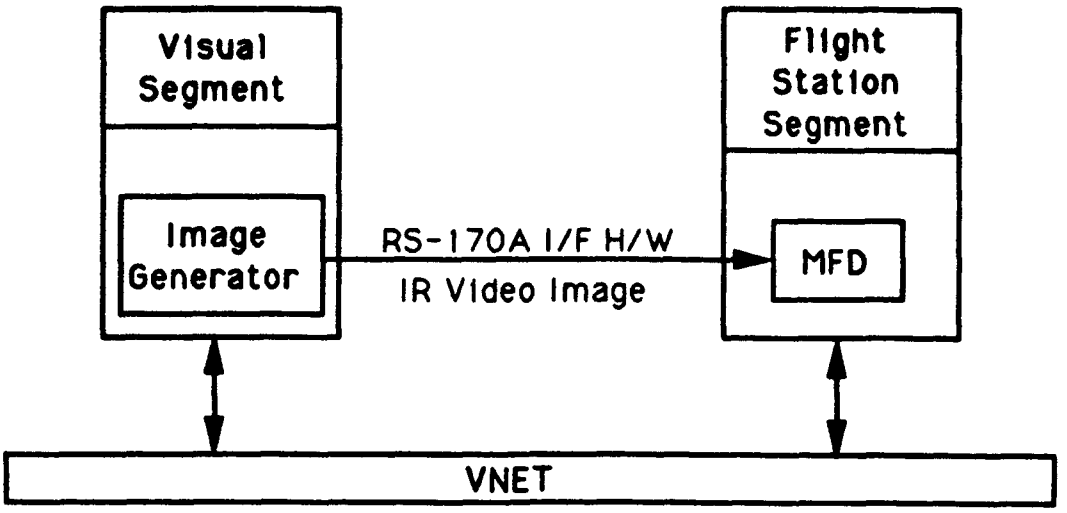
Figure 4.2.4-1 provides an example of a typical back door interface and an example of a misuse of a back door interface. Back-door interfaces are an exception to the rule for Mod Sim and must be carefully considered before allocation.

4.3 Hardware Requirements. The specific computational hardware applied to a simulation program should be consistent across the system, although it is not a firm requirement as hardware may vary from segment to segment within a system based on specific program needs. Common hardware decreases acquisition and life cycle maintenance costs. Acquisition costs are almost always reduced because of the discounts associated with large quantity buys hardware and spares. Life cycle maintenance costs are also reduced because of less unique support equipment and lesser requirements for unique maintenance skills. Regardless of the advantages, the use of common hardware must not sacrifice performance requirements unless trade studies indicate otherwise.

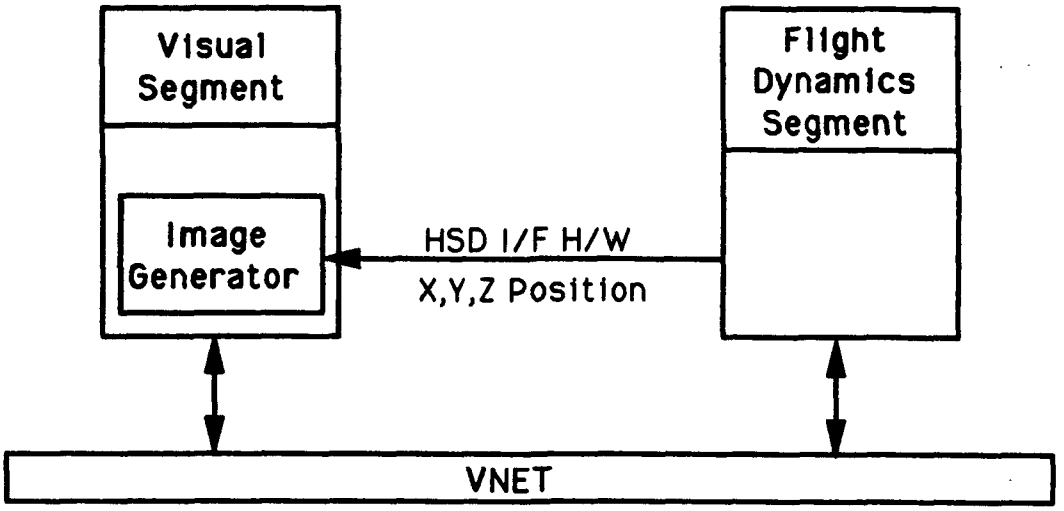
For a Mod Sim, the software architecture is a key factor in selecting a hardware solution. When selecting the hardware solution, one should have a general concept of the hardware design, but one should also be careful to not lock in prematurely. Mod Sim is fundamentally a software architecture which is adaptable to many different hardware applications and therefore the selection may be made later in the program. This will afford the opportunity to take advantage of performance increases associated with the rapidly changing computational hardware market.

4.3.1 Computer Architecture. The following guidelines are recommended in selecting a computer architecture for the application Mod Sim.

4.3.1.1 Computational Hardware Interrupts. The interrupt requirements for the application CPUs must be defined prior to selecting the hardware for each segment or module. Interrupts allow synchronization of segment and module execution. In Mod Sim, the Clock_Tick message is a high priority interrupt for this express purpose. Since other hardware devices communicate with processors, through interrupts, consideration must be given to ensure that the number of interrupts meets simulation requirements.



Typical Back Door Interface



Prohibited Back Door Interface

Figure 4.2.4-1
Back Door Interface Examples

Also the priority vector of the interrupts must accommodate all hardware elements (e.g. VNET CPUs, FDDI boards, etc.) in the Module.

4.3.1.2 Byte Transmission Architecture. A significant consideration in selecting computer hardware for an application is byte order of transmission. The preferred architecture is most significant byte to least significant byte, also known as Big-Endian architecture. This implementation simplifies the interface with the VNET, which is currently defined in accordance with MIL-STD-1777. If the processors selected are least significant to most significant order, Little-Endian architecture, there will be a requirement for the processors to translate between the two formats, if the implemented VNET is Big-Endian. A final consideration in selecting one or the other architecture is potential reusability. Since Big-Endian is more prevalent, it follows that there should be more opportunity for reuse with that implementation.

Regardless of whether the processors are Big or Little-Endian, it is recommended that they be the same across the simulation system. If they are not, then the majority type processor must be determined, and a representation specification prepared for a Presentation Layer routine (see Section 7.1.7) that swaps bytes to execute on all minority CPUs for virtually every message. Alternatives to this Presentation Layer routine are available. For example, it is possible to pass the byte-order requirement to the network interface so that explicit Presentation Layer translation is not required.

4.3.2 Network Interface Hardware. One of the most important trade studies to be performed on a Mod Sim is the trade to determine the network interface hardware. The factors to be considered include required data rate based on the tailored message content, iteration rate, segment to module allocation, flexibility to reallocate segments (i.e. growth), standardization, and transmission reliability. Since the interface hardware is the prime element for standardization in a Mod Sim, the effects the selection will have on the individual segments and modules is probably the overriding consideration of the study.

4.4 Software Architecture. Software architectures for all segments should be defined during the Systems Engineering phase of the program. Two alternatives exist in determining a system's software architecture. A unique software architecture for each individual segment that best suits that segment's requirements or a common software architecture for all segments in the system. The advantages of the latter include potential reusability and the associated reduction in software development costs, simpler integration, increased design standardization, and others. Since the performance to cost ratio of processing resources tends to make quantum increases, an increasingly disproportionate percentage of program costs are associated with software development. Reusable software offers at least a partial solution to minimizing program development costs.

Although a common architecture is generally preferable, it does have some disadvantages. A common architecture will undoubtedly constrain the segment developers to some degree by narrowing the developer's choice of hardware and/or software solutions. It can actually eliminate some innovative solutions altogether. A common architecture may even eliminate potential segment developers due to

inexperience with the architecture or product incompatibility. In general, the greater the number of segment developers, the greater the impact of a common architecture. Conversely, a program with few segment developers affords an excellent opportunity to standardize the segment software. Section 5 presents a more detailed discussion of software design considerations.

4.4.1 Operating System(s). One factor in the selection of the application hardware is the operating system (OS) associated with the hardware. The OS must support all hardware and software interrupt requirements for the simulator. In addition, VNET interface processing must be able to interrupt real-time simulation processes in order to provide simulation synchronization.

4.4.2 Frame Timing. Probably, the most important trade study to be performed on any simulator is the one that ultimately determines the device frame rate. As stated in Section 4.1.6, timing requirements are a key factor in this analysis. In addition to latency requirements, subsystem interoperability requirements may influence the frame rate. When actual aircraft avionics hardware and software are used in a device, it may be necessary to use the aircraft frame rate, or sub multiple thereof, to synchronize the simulation to the avionics. If avionics compatibility dictates a different frame rate than the timing analysis, it may be necessary to relax timing requirements, identify more complex alternatives or reconsider stimulation of avionics components.

4.4.3 Data Engineering Units. In a modular simulator architecture, communication of data between segments is expressed in engineering units according to the Appendix A interface definition. A message recipient is required to make the proper engineering unit assignment to each variable and convert to internal segment units, if necessary. The segment designer is responsible to adhere to the Appendix A engineering unit assignments and to perform internal calculations concerning individual variables at the assigned resolutions. This will allow using segments to make proper assumptions as to the accuracy of the variable data values.

5. SOFTWARE DESIGN ACTIVITIES

As part of the top level system design activity, Systems Engineering defines the system architecture, including VNET communication protocol, segment/module configuration, the required functionality for each of the segments, and the system level interface requirements. Segment commonality trade studies may warrant additional system requirements such as a common VNET interface for all segments, common software language, common segment software architecture, common executive functions, and others. This section addresses segment software architecture, VNET interfaces, coordination and communication between segments, messaging, software coding, and software testing as they relate to a Mod Sim.

5.1 Segment Software Architectures. In order to preserve one of the premises of the Mod Sim concept, internal segment design is not normally mandated. This allows open competition for segment proposals from a broad base of the simulation industry. However, trade studies may determine that some measure of commonality can be beneficial and in those cases Systems Engineering will specify the software and/or hardware elements which must be common across the segments. As previously stated in paragraph 4.4, common architectures offer many benefits over segment specific architectures. It is recommended that system designers consider common software architectures for all segments in the system.

Another factor concerning common segment architecture is the emergence of domain engineering. It is expected that future application programs will implement common Domain Specific Segment Architectures (DSSA), to take advantage of the reusability offered by both the Mod Sim concept and domain engineering. Even though the DSSA is expected to become the norm for Mod Sim design, inevitably some programs will be exceptions, and will require one or more segments to have special software architectures. DSSA is discussed in more depth in Section 6.

5.1.1 F-16C Mod Sim Segment Software Architecture. In the Mod Sim F-16C Demonstration program, segment software designers were given segment interface and timing requirements as their only design requirements. As a result, segment software architectures differed somewhat. Different software languages were used, translators were used, and some software was used as is.

All segment designers adopted a software architecture based on the same sample executive (cyclic) program. Beneath the executive level, however, two different data control design approaches were taken. In the first design, data sharing between functions was implemented by storing the data in a common memory area, allowing all software elements to access this data at will. In the second design, data flow was controlled, isolating the inputs and outputs of each function and passing the data in function calls. These segments proved to be easier to debug and integrate since reading and writing of data was explicitly defined, and therefore, it was determined that this architecture would be easier to adapt and reuse. The following guidelines will aid in implementing this design:

a. Data objects may only be declared in Ada packages, tasks and subprogram bodies. In particular, this prohibits creating an Ada package called Global which contains all the variables used to communicate between subprograms.

b. Segments may be decomposed to the depth required for the program (Figure 5.1.1-1). Data may only flow from one element to another through the subprogram parameters for each element.

Assume that Object_1 has an output (called X) that is required by Object_4 as one of its inputs. Further, assume that each of the Subsystems and Objects are called left-to-right: The Segment Scheduler calls Subsystem_A, then Subsystem_B, and finally Subsystem_C; Subsystem_A calls Object_1, then calls Object_2, and so on.

In this software architecture, Object_1 is called with X as one of its outputs. X is also one of the outputs of Subsystem_A. X is an input to Subsystem_C and is an input to Object_4. This data flow is illustrated in Figure 5.1.1-2.

c. Considerable clarity can be gained if all the data flow is localized at one level. Accordingly, all communication is located near the top of the segment software structure, in the Segment Scheduler. Pseudo code for the scheduler would look something like the following:

```
Get_The_Messages_From_The_Virtual_Network(Function_Ins);
Execute_All_The_Functions(Function_Ins, Function_Outs);
Send_The_Messages_To_The_Virtual_Network(Function_Outs);
```

5.1.2 VNET Interface. Paragraph 4.2.3 introduced the VNET and its top level requirements. The following paragraphs provide additional detail with respect to its functional requirements and top level design.

5.1.2.1 VNET Interface Performance Requirements. Two essential requirements exist for the VNET Interface. They are reliability (data integrity and error handling) and speed (communication response). The VNET interface must manage routine transmission errors, and notify the Application in the event of an unrecoverable error. The VNET must complete the transmission of a message within the required time, to minimize Application time spent waiting for VNET response to its requests.

5.1.2.1.1 Data Integrity. The Application should control the transfer of data from the VNET. The VNET Interface should not write into the Application's data area except at the request of the Application. The Application should call a function to command the VNET to transfer data to it. Between these calls, the Application must be assured that no data transfer will take place.

There is always a possibility that the VNET Interface might be in the process of reading in a message at the very time that the Application asks for that message. Since the message is incomplete, the VNET Interface should not make the message available until the whole message has been transferred in from the VNET.

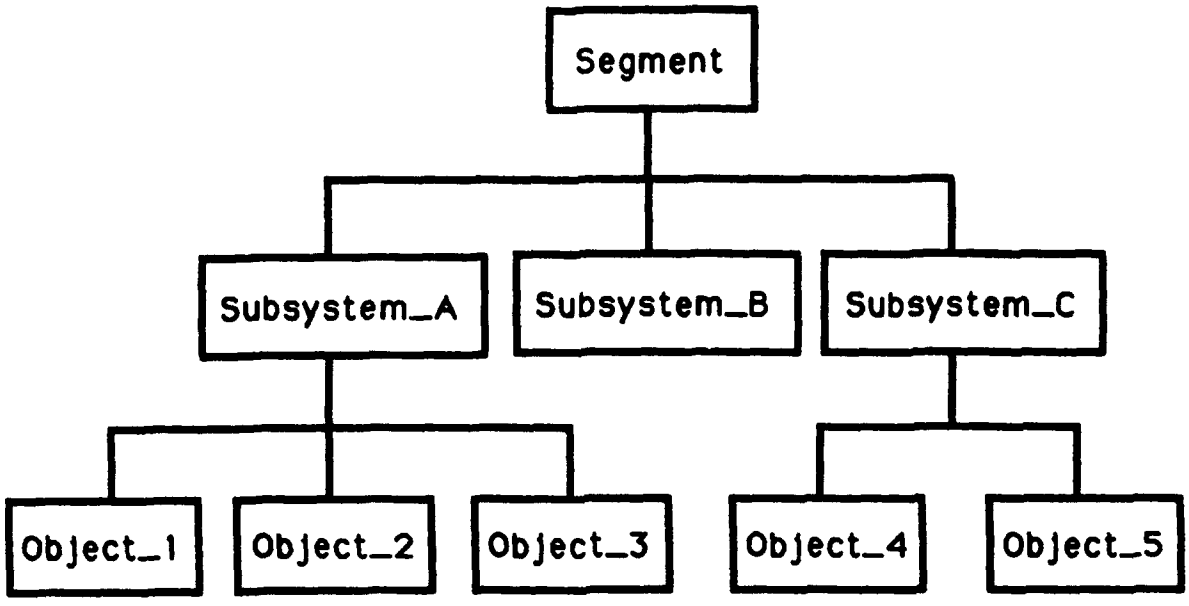


Figure 5.1.1-1
Sample Segment Architecture

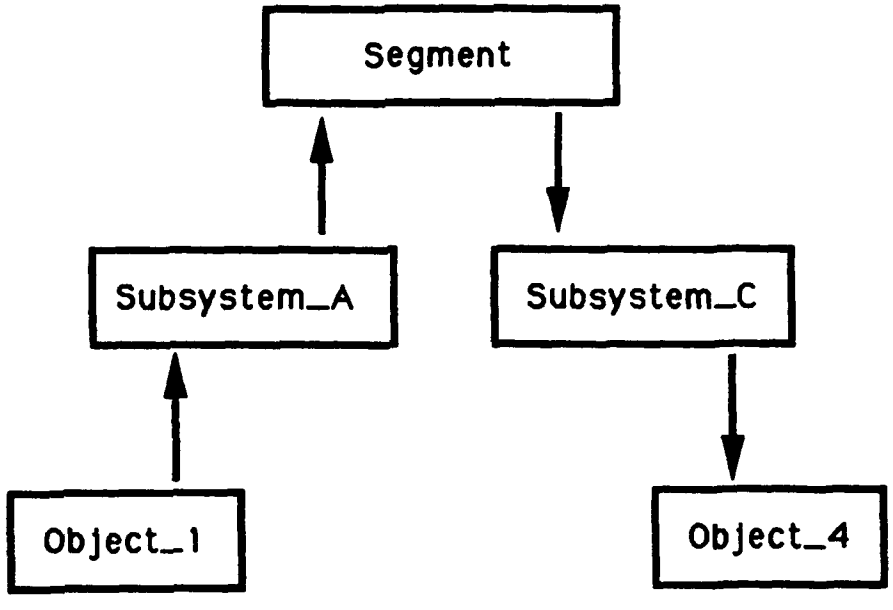


Figure 5.1.1-2
Data Flow from Object_1 to Object_4

Another data integrity requirement is that data, once sent to the VNET Interface software, cannot be modified or corrupted en route. That is, the interface is responsible for the proper transmission of the data once the Application requests a message transfer to the VNET.

5.1.2.1.2 Error Handling. Every medium for data flow between segments whether it be shared memory on the same CPU, shared memory on a back plane, or a network bus like FDDI, carries the possibility of error. It is essential that the VNET Interface not present erroneous or incomplete data to an Application. The protocol specified for the VNET on the Mod Sim Demonstration program was the XTP. XTP met the requirement for error detection and automatic retransmission of data at a very low cost in computational overhead.

When responding to a request for data, the VNET Interface should notify the Application when either no data or no new data exists, so that the Application can take the appropriate action. The first condition arises when a message of a given type has never been received by the VNET Interface. A probable cause of this condition is that the sending segment is not connected to the VNET or a change in the original state of a send on change message (See paragraph 5.3.2) has yet to occur. In these instances, a buffer or memory location associated with the message will already exist, but the Application should not use the "data" in it. The VNET Interface should always alert the Application when the data should not be used.

The second condition arises when no new data has been received by the VNET Interface. For example, assume that the Application has asked the VNET Interface for a specific variable, and the VNET Interface has provided the data. After some time passes, the same Application asks for the data again. If the VNET Interface has not received another message of that type, it must alert the Application that the data has not changed. This is especially relevant for send on change messages. Malfunction insertion, for example, consists of an Application query to the VNET Interface, to find if there is a subsequent malfunction message to process. If there is, the VNET Interface delivers it.

A third condition can exist where new data is available but it is erroneous from the sending application. In this case, the interface has no responsibility for error detection or retransmission. Range checking and other detection methods must be the responsibility of either the sending or receiving segments or both.

5.1.2.1.3 Communication Response. There are two performance factors associated with message transmission speed. First, there is the fundamental system level requirement that all messages transmitted in one frame must be available to the destination Application(s) at the beginning of the next frame. Since a message can be transmitted at the last possible moment in the usable portion of a frame, the allotted time is simply the minimum spare time requirement specified for a frame. For example, in a simulator with a 50% spare frame time requirement, an Application can command the VNET Interface to send a message as late as halfway through the frame. This translates into a system level, half-frame requirement for end to end transmission of all messages in a given frame. With this system level requirement as the baseline, the

segment designer must then address the second factor, which is VNET interface response time. The segment designer must determine the maximum amount of message traffic, in and out, for the worst case frame of the segment to determine how fast the VNET interface must operate for the largest transmission requirement.

5.1.2.2 VNET Interface Functional Requirements. A VNET Interface, as defined in the Mod Sim design, is required to send and receive messages to and from the network. The following paragraphs describe the required functions for a VNET Interface.

5.1.2.2.1 Send Message and Send List Functions. These VNET functions send data to the VNET in response to Application requests. While in transit to the VNET Interface the data should be protected from writes by the Application. In the event of an error, these functions should automatically and transparently retransmit the data. If a catastrophic error occurs, the functions should notify the Application accordingly. These functions should also return the status of the transmission, which is either successful or the VNET Interface has determined that the segment is an unauthorized sender for the message or list of messages.

5.1.2.2.2 Receive Message and Receive List Functions. These functions makes data available to the Application upon request. They makes only complete copies of a given message available to the Application and return the status as either no data, no new data, new data or the segment is an unauthorized receiver for the message or list of messages.

5.1.2.2.3 Segment Identification Function. When common VNET interfaces are used, this function identifies the segment to the interface so that the interface can determine which messages the segment is authorized to send and receive.

5.1.2.2.4 Message List Function. This function identifies each of the messages, and the authorized senders and receivers for each message.

5.1.2.2.5 Number of Copies Function. This function identifies the number of copies of each message type to be kept by VNET Interface to ensure that required messages are not overwritten. This is particularly important for send on change messages of the same type (Section 5.3.2 discusses multi-copied messages).

5.1.2.2.6 Application Interrupt Function. This function interrupts the Application layer upon receipt of specific messages which require immediate segment action. The most notable example is a segment synchronization message normally issued by the IOS segment. Others may include send on change messages.

5.1.2.3 Additional VNET Interface Design Considerations. The following paragraphs address additional design topics and some lessons learned with the Mod Sim Demonstration program.

5.1.2.3.1 Communications with the VNET. The means of communicating with the VNET is dependent on the hardware and operating system used on the system. Therefore, the design of the VNET Interface must be considered in the design trade studies, as the hardware and operating system are selected.

Because the VNET must respond to multiple requests, there must be a means of communicating the requesting segment, the request, and data associated with the request (e.g., Message ID, Status Return) between the segment Application and the VNET. On the Demonstration program, each segment had its own VNET process. The segment wrote to a known memory area and generated a VME bus interrupt for the VNET process.

On a VME bus based system with each VNET Interface in its own CPU, an area of memory may be allocated to control blocks, one per segment, where the segment can write request data. The segment CPU interrupts the VNET CPU, and the VNET CPU scans the request areas and processes all requests .

On a Unix-based system, the segment Applications may use Unix pipes to send request data to the VNET. The VNET process would then "pend on the pipe" or respond through a signal handler.

5.1.2.3.2 VNET Data Control. In a Mod Sim software implementation a data object is declared for every message. These declarations are in packages called:

<*segment name* >_Output_Interfaces

There is one package per segment, and these packages are stored in a directory which contains nothing else. Comment fields near the message object declarations in these files define all the other segments to which the message is to be sent.

On the Demonstrator program, a utility program went through each of these files and built an Ada package consisting of the message name, declared as an enumeration, and an array of 32-bit integer flags indexed by that enumeration. The message name was copied from its declaration in the message object files. The flags were bit flags constructed from the comment fields. For example, if the Environment segment had the enumeration value of 1, the 1 bit would be set for any message destined for the Environment segment. The 32-bit flag was divided into two 16-bit halves. The first half defined the authorized sender, in which only one bit would be set, since only one segment is authorized to send a given message. The second half contained a bit map which defined the authorized receiver(s) of the message.

There were three problems with this approach. First, comment fields were used to define the authorized receivers of the messages. It is a questionable practice to rely on comments, since they may not be accurate. Second, the enumeration and bit fields had to be compiled into the VNET software. Each change in the definition of a message resulted in a requirement to recompile the VNET software. Third, the declaring of data objects in a package is contrary to some recommended coding standards. In fact, the message object declaration files were used for no other reason than as inputs to these utilities.

It is a reasonable alternative to put the message names and the authorized sender and receiver information into a data file which is read at initialization by all VNET Interfaces.

The use of data file eliminates the need to recompile every time a message destination is changed or added.

5.1.2.3.3 Buffer Allocation. Data buffers for storing message data which the individual segment Application can access must be allocated. On the Demonstration program, these buffers were allocated to the VNET CPU. The VNET Interface, when queried, merely returned the address of the buffer containing new data to the Application. An alternative design may move all copies of a message from the VNET to the Application's memory on a Receive Message function call. Then, when the Application required the next copy of the message, the Application Services (a Mod Sim implementation discussed in Section 5.1.3) would advance to the next copy without accessing the VNET.

On the Demonstration program, some of the segment hardware devices, when combined with the traffic from the FDDI board, could cause VME bus time-outs. In order to resolve this problem, the VME bus time-out interval was increased. This experience shows that designers must consider the memory bus in locating message buffers. For example, if the Application CPU board has the ability to perform fast DMA transfers, and if this will not interfere with other requirements of the back plane, locating the message buffers in Application memory may be desirable.

Finally, it is possible that segment Applications will execute in memory that will not be visible to the VNET CPU, but that the VNET CPU's memory will be visible to the Applications CPU. In this case, the Application Services must perform data transfers. If the opposite is true, and the Application memory is visible to the VNET Interface, the VNET Interface must perform the data transfers.

5.1.2.3.4 Network Drivers. The VNET Interface software must also manage network communication hardware. This software will include network drivers (e.g., FDDI board drivers) for the appropriate hardware and operating system, and auxiliary software (e.g., sockets) to communicate with the network driver.

5.1.2.3.5 Interrupts from the VNET. The implementation of the Application Interrupt function will depend on the application hardware and operating system. On the Demonstration program, the segment Application determined the subprogram to be executed within its Application Services when a given message arrived. The Application Services, in turn, informed the VNET Interface to interrupt the Application processing whenever the message was received. When the message arrived, the VNET software placed the Message ID in a known location and generated a VME bus interrupt. Application Services, which contained an index of Message IDs to subprograms, then executed the proper subprogram.

5.1.2.3.6 Starting and Stopping. Some of the activities of the VNET Interface (e.g., Send and Receive Message functions) must be executed as quickly as possible to reduce the effects of latency, while others, such as converting the message name and allocating message buffers, might be time consuming, and might even interfere with real-time operation. The Demonstration program had a function called Start_BIU which served as the dividing line between real-time and non-real-time operations. Attempts to perform real-time functions prior to completion of Start_BIU were considered to be error

conditions. For example, messages received from the VNET before the call to Start_BIU were ignored and discarded, since the message buffers were not allocated yet. Similarly, attempts to execute non-real-time functions after the VNET Interfaces were initialized were considered to be errors.

There was no Mod Sim requirement to stop and restart real-time operations. If this requirement exists for an application simulator, Start_VNET and Stop_VNET services must be implemented. These services will place the VNET Interface in the proper state for performing real-time and non-real-time operations respectively.

5.1.2.3.7 VNET Interface Response. Software Engineering must decide during software preliminary design about waiting for the VNET Interface to respond to a request. In the Demonstration program, the segment Applications waited until the VNET software finished processing the request. This design was implemented due to the requirement that Application data in messages must be immediately readable after a Get command and must be immediately writable after a Put command (Get and Put commands are discussed under Application Services later in this Section). In this implementation, the Application is blocked while data is transferred to or from the VNET, it writes to the buffer immediately after a Put command, and reads from the buffer immediately after a Get command. The VNET software responds "done" as soon as it has copied the data, even though it may still be processing the data. It is important that the VNET Interface respond as quickly as possible without allowing data to be corrupted.

5.1.2.3.8 Presentation Layer. As discussed in paragraph 4.3.1.2, the Presentation Layer may have to translate between Big-Endian and Little-Endian representations and translate various floating point formats between the Application and the VNET. On the Mod Sim program, the Presentation Layer was resident in the VNET software.

In some applications, CPUs of different architectures may reside on the same back plane. In this case, the Presentation Layer reformatting must be performed in the Application Services. Otherwise, it is more convenient for the VNET software to perform this reformatting for all its client CPUs.

5.1.2.4 VNET Interface Software Reuse. A primary goal of modular simulation is software reuse, and in particular, the VNET Interface software should be reusable between segments. It should be possible, given identical computational hardware, for the same VNET Interface software to operate with every segment in a Mod Sim. Nothing about the VNET Interface should be specific to a given segment.

5.1.3 Application Services. On the Mod Sim Demonstration program, Application Services provided access to the VNET interface for the segments. In cases where there are several segments in the same module, the Application Services must service each segment without introducing latency. This may require several instantiations of Application Services if the message traffic warrants. The following paragraphs describe the various Application Services functions.

5.1.3.1 Put and Put_List Functions. The Put function is used to send a message to the VNET. It is recommended that the Put function be called for a message as soon as all

the data in the message have been computed. Using the Put function in this manner distributes message traffic throughout the frame. If segments waited until the end of a frame to call Put, all the traffic on the VNET would be concentrated at the end of the frame. Put_List is identical to Put except a list of messages is sent to the VNET rather than a single message.

5.1.3.2 Get and Get_List Functions. The Get function requests the VNET to provide a new copy of the desired message. Get_List permits the VNET Interface to poll and update the status of an entire list of messages in one call. The Mod Sim Application program interrupted the VNET software for every Get_List request. The Get_List call was created to reduce the number of interrupts experienced by segment VNET CPUs. The time required to respond to a large number of software inquiries would overload the VNET CPU for some of the segments. Since the greatest number of requests a typical segment made were Get calls, and since Get calls were cyclic, (i.e., the same Get calls were used at the start of every frame 1, every frame 2, etc.) a Get_List function was found to be beneficial.

5.1.3.3 Connect_To_VNET and Disconnect_From_VNET Functions. When connected to the VNET, all messages to and from the segment are processed by the VNET interface. When disconnected from the VNET all messages to and from the segment are ignored.

5.1.3.4 I_Am Function. The I_Am function identifies the segment to the VNET interface to ensure that the segment is an authorized sender for output messages and an authorized receiver for input messages. The only reason for this function is to allow the use of common VNET interfaces across the device. If unique interfaces are used, the segment identification could just as easily be hard coded in the interface.

5.1.3.5 Define_A_Message_Record_For Function. This function defines a message record, attaches it to a message buffer and defines the communication mode of the message (i.e., Put or Get). It is an initialization function which is called once for every message type intended to be transmitted or received by the segment.

5.1.3.6 No_Operation Function. This function probes the VNET interface to determine if the interface can respond. It is an initialization function which is executed repeatedly until communication is established.

5.1.4 Application Software. The Application software consists of the segment support services and the simulation functions. With respect to the simulation functions, the segment developer should be afforded maximum flexibility in defining the design. Conversely, the support services, and in particular the executive service, are excellent candidates for reuse and therefore should probably be specified to be common across the simulator segments.

Another issue concerning segment software is the communication between simulation functions. In keeping with the concept of a Mod Sim, communication between functions within a segment is typically at the discretion of the segment developer. Section 4.4, however, discusses the advantages of common versus segment specific architectures. The same arguments would probably apply to internal segment communications.

5.2 Segment Synchronization. Every segment is required to produce and provide to the VNET certain messages at certain rates. It is therefore necessary to synchronize the processing of all segments to a common clock.

5.2.1 Segment Synchronization Message. A segment synchronization message, sent by the IOS to each segment, provides the common clock. Each segment executive function must allow the VNET Interface to interrupt the segment whenever this message arrives. Therefore, even though segments may be distributed on separate processors, their activities must be coordinated through some form of synchronization message.

5.2.2 Simulation Frames. Upon receipt of a segment synchronization message, each segment begins a new frame (regularly scheduled, constant time intervals in which segments must execute a predefined list of tasks prior to starting the next frame). During each frame, a segment executes a specified portion of its iterative functions and sends the appropriate messages (see Sections 5.3.1). A segment must send maximum rate messages each frame, half rate messages every other frame, and so on. For example, Flight Dynamics sends the messages:

1) Equations_Of_Motion_Max_Rate every frame,
2) Companion_Vehicles_Half_Rate every other frame,
3) Equations_Of_Motion_Quarter_Rate every fourth frame
4) Weight_And_Balance_Eighth_Rate every eighth frame.

The frame rate assignments given in Appendix A of the IDD, are based on previous experience with simulators and were coordinated during the Mod Sim program. These assignments were based primarily on simulation fidelity requirements. For example, weight and balance data changes by such small increments that it only needs to be sent every eight frames, while equations of motion data changes so rapidly that it must be sent every frame.

Systems engineers commonly decide that the simulation need not be broken down any farther than eighth-rate or sixteenth-rate. It is conceivable, particularly when the maximum rate is very fast (100 Hz or more), that 32nd rate messages may be allocated. In this case, many of the maximum rate messages in the interface specification will be lowered to half-rate, half-rate messages will be lowered to quarter-rate, and so on.

The IOS will set a simulation frame number in the segment synchronization message, depending on the iteration rate chosen for the simulator. For example, if sixteen Hertz is the highest iteration rate for a message, the IOS will count frames 1 through 16, then start over again at frame 1.

5.2.3 Segment Scheduler. This Mod Sim executive service examines the contents of the segment synchronization message and executes the appropriate set of segment Application programs based on the current frame. This is commonly handled in a Segment Scheduler subfunction of the executive service Pseudo code for a Segment Scheduler may look like the following:

```
loop
    case Clock_Tick_Message.Current_Frame is
            when 1 =>
                    Get_The_Frame_1_Messages_From_The_Virtual_Network;
                    Execute_The_Frame_1_Functions;
                    Send_The_Frame_1_Messages_To_The_Virtual_Network;
            when 2 =>
                    Get_The_Frame_2_Messages_From_The_Virtual_Network;
                    Execute_The_Frame_2_Functions;
                    Send_The_Frame_2_Messages_To_The_Virtual_Network;
                    ....
                    ....

            when n =>
                    Get_The_Frame_n_Messages_From_The_Virtual_Network;
                    Execute_The_Frame_n_Functions;
                    Send_The_Frame_n_Messages_To_The_Virtual_Network;
    end case;
end loop
```

This Segment Scheduler implementation will satisfy the requirements for coordination within the segment.

During preliminary software design, functions and messages should be allocated to frames. For example, if function A computes a temporary result which is used by function B, the delay between A's inputs and B's outputs will be minimized if the two functions execute in the same frame, with function A executing before function B. Data flow analysis will assist in this allocation. The initial allocation of functions to frames should be flexible, since frame balancing may be required during the segment integration phase.

5.2.4 Coordination Between Segments. During Preliminary design, functions should be allocated to frames to minimize latency between segments. For example, segment A sends a quarter-rate message to segment B, which uses the data to compute other data, which it sends in a quarter-rate message to segment C. Segment C uses the message to compute data, which it sends in a quarter rate message to segment D, which displays the result. Latency between segment A and segment D will be reduced if segment A sends its messages on simulation frame numbers 1, 5, 9, and 13; segment B sends its messages in frames 2, 6, 10, and 14; and segment C sends its messages in frames 3,7,11,15. Function and message frame assignments should minimize the throughput delays in the simulator as a whole.

In order to coordinate between segments, it is necessary for the executive function of a segment to provide notification when it falls behind. If this coordination is not provided, and a segment is not able to complete its work before the next segment synchronization message, it will send its output messages in the wrong frame, and segment functionality will probably suffer.

Accordingly, the segment executive must have some means of providing detection of this condition. When this overrun condition is detected, the segment may:

1) flag the overrun by notifying the IOS to record and display the condition
2) increment an overrun counter
3) attempt to catch up with no indication outside of the segment
   or
4) halt the simulation with an error message.

Requirements for the particular training system will determine the method of handling overruns. These requirements should be based on the consequences of an overrun (e.g., safety, loss of fidelity, data integrity, etc.).

A common method of overrun detection relies on incrementing a counter in the segment common memory when a segment synchronization message is received. The segment executive service picks up this counter at the beginning of the frame. At the end of the frame, the segment executive function compares its counter to the current counter in common memory. If the two are different, an overrun has occurred. An example of executive function pseudo code that will perform this function is:

```
loop
        This_Frame := Common_Memory.Current_Frame;
        Execute_One_Frame;
        if This_Frame /= Common_Memory.Current_Frame then
                Notify_Of_Overrun;
        else
                Wait_For_Next_Clock_Tick;
        end if
end loop;
```

5.3 Messages. Mod Sim communicates between segments through the use of messages. There are two types of messages; iterative and send-on-change. This section discusses these message types and some guidelines and considerations concerning them.

5.3.1 Iterative Messages. An iterative message is a message which a segment sends at regular intervals while the simulator is running. Typically, iterative messages contain simulation variables which represent real world conditions that are continuous in nature. To approximate continuous real world functions, the corresponding simulation functions are executed iteratively at rates sufficient to prevent the air crew from perceiving the resulting stepping functions. The iteration rate for a given message is determined by several factors. These include coupling requirements to dependent functions, function criticality, display rates for those message variables representing crew station display values, cue correlation and transport delay requirements, change rate of a variable, to name just a few.

5.3.2 Send-on-change Messages. Unlike an iterative message, a send-on-change message is sent only when a change in the message data content occurs. Normally the variables contained in send-on-change messages are discrete in nature and not subject

to frequent change. The primary reason for a send on change message is to reduce redundant message traffic on the network and the time associated with receiving them. Another reason for send on change messages is that they may have an interrupt associated with them requiring unique segment response. For example, the Instructor/Operator Station (IOS) may use a send on change message to inform a segment to change simulation states.

Special care must be taken by segment designers to avoid overwriting the first send on change message status with a second. The VNET Interface must be able to store multiple copies of a message and present them to the Application. For example, an automated IOS page responds to an instructor selection by setting Malfunction_A and then clearing Malfunction_B. Each of the malfunction messages is of the same message type. If the VNET Interface for a segment that receives the malfunction messages has space in its buffers for only one copy of a message, it would overwrite the message that arrived first (the Malfunction_A message) with the message that arrived second (the Malfunction_B message). As a result, the segment Application code would not know to set Malfunction_A.

The approach employed on the Mod Sim Demonstrator for handling multi-copied messages was to send a variable-length list. The message would contain data on a list of threats, for example, along with the number of threats that were currently active. That number may range from zero to the maximum number of threats permitted in the simulation. One message of this type could be sent each frame, even if there are no active threats.

The Mod Sim Demonstrator was limited to only a few active threats, so that the occurrence of multi-copied messages was limited. However, in a simulation that requires a large number of active threats at one time and a lot of data associated with each threat, the message containing all active threats can be quite large. These large messages pose problems for some VNET implementations. Accordingly, a different method has been adopted for the current Mod Sim architecture.

The current Mod Sim architecture allows multiple copies of a message to be sent during a frame. This permits an Application to send one message for each threat that it controls. In some circumstances, when there are no threats, no message will be sent in the current frame. In other cases, several copies of the same message, each representing a different threat , will be sent in a given frame.

5.3.3 Hardware Considerations for Maximum Message Length. When a message is longer than the Maximum Transfer Unit (MTU) of the media or the protocols, it must be broken into pieces by the sender and reassembled by the receiver. This reassemble process is very time-consuming.

To mitigate this time penalty, the designer must assure that the largest message will fit into one packet on whatever media is used. The maximum allowable length is computed as follows:

$$\text{Max Message Length} = \text{MTU} - \text{PL} - \text{ML} - \text{GF}$$

where;
MTU is the smallest maximum transfer unit for the network(s) that may be employed on the simulator to transmit the message (If more than one network type will transmit messages, the shortest MTU should be used);
PL is the length of the longest protocol header which may be used;
ML is the length of the longest Media Access Control (MAC) header which may be used;
GF is a growth factor to allow for the growth of messages to incorporate new data.

Note that if there are protocol or MAC trailers, these should be added to PL and ML, respectively.

If the VNET is implemented as shared memory throughout the system, MTU is not limited. However, simulators often transfer data over Ethernet, for example, to the IOS for display pages. ETHERMTU (1,500 bytes) may be used for MTU, but this value must be used with caution, since some custom network drivers have smaller buffers.

Protocol and MAC headers and trailers are specified in the protocols and interface hardware documentation. A factor of 100 bytes for the sum of these headers and trailers generally offers a reasonably safe growth factor for message growth.

5.3.4 Assigning Messages to Frames. Messages are assigned to frames in the Mod Sim based on several factors. One factor is CPU load balancing. The proficient segment designer should attempt to balance the frame time load on the CPU by distributing the execution of functions among the frames. Frame balancing is considered good design practice because it affords the most flexibility to incorporate new functions. If the function that produces a message only executes every 4 frames, the message only needs to be sent every 4 frames. Another factor is coordination between segments. Delays in computation due to message traffic must be minimized throughout the simulator. A final determining factor is message traffic balancing. Within the constraints of load balancing and coordination, the segment designer should attempt to send roughly the same number of messages every frame, to avoid too much traffic on the VNET. This is especially important when the VNET is implemented as an actual hardware network.

During preliminary design, messages should be provisionally assigned to frames The designer should make clear notes of the logic behind message frame assignments, so that as actual execution times and network loading become known, necessary adjustments can be made considering that logic.

5.3.5 Message Names. New messages required for the Application simulator should be defined in the software prior to or during preliminary design. It is recommended that the message naming convention used on the Mod Sim program be adhered to, in order to take full advantage of Mod Sim reusability and utilities. The message related utilities used in the Mod Sim Demonstration program relied on the names given to the messages. For example, a message labeled "xxx_Quarter_Rate" was sent at quarter rate.

5.3.6 Message Identifiers. An identifier should be associated with each message. On the Mod Sim Demonstration program, the Application software supplied the message name to the VNET interface, and the VNET interface returned a Message ID (a unique integer) to the VNET. The Mod Sim software should also contain a segment identifier, since the segment Application must identify itself to the VNET.

5.3.7 Authorized Senders. The Mod Sim architecture dictates that a given message may be sent by one and only one segment, as defined in Appendix A of the IDD. The VNET Interface is responsible to assure that messages are sent by the authorized segment. Therefore, the VNET Interface must be able to notify the Application when it is attempting to send a message for which it is not a legal sender. Likewise, the VNET Interface (in keeping with the reuse requirements above) must be capable of identifying legal senders of messages to the VNET.

Assigning messages to a sender must be completed before software engineering activities start, since messages are associated with functions which must be assigned to segments during system design phase. Segment designers may continue to define message receivers during software design, as they determine requirements for data flows.

5.3.8 Authorized Receivers. The Mod Sim architecture also requires that the authorized recipients of a message be identified, as defined by Appendix A of the IDD. A segment is not allowed to receive a given message unless it is a authorized receiver of the message. Due to this requirement, the VNET software must contain a list of legal receivers for each message.

5.4 Engineering Test. Software Engineering test in a Mod Sim consists of stand alone segment testing, to assure that each segment performs in accordance with its functional requirements and that it has a high probability for successful integration.

5.4.1 Segment Test. A segment test tool is required to perform stand-alone segment tests. The primary purpose of the segment test tool is to test the external interfaces of each segment , defined in Appendix A of the IDD, that resides on a given computer platform. It is recommended that designers of the segment test tool base the tool design on the Appendix A interfaces. This design will allow the segment test tool to be easily updated throughout the program as the Appendix A interfaces are modified.

It is further recommended that input test data be developed using user-friendly construction tools that utilize standard Graphical User Interfaces (GUIs) to build windows and pages. The input test data should be checked for correctness against the interface specification. The user should have predefined interface data available for use or change as required by specific tests. In addition, the user should be able to selectively record test results data by selecting the interface data that is of interest and only recording that data. This will aid in the evaluation of the test results. Segment testing can be divided into three distinct parts:

a. Test data file construction

b. Segment test control

c. Output test data formatting

5.4.1.1 Test Data File Construction. The test data file construction tool should provide options to create new test data files, load, copy, or delete existing data files, save created or modified data files, and print data files. Use of the test data file construction tool should follow the steps defined below:

a. Select a file creation/change option

b. Input a file name.

c. Input the segment to be tested

d. Define input messages and associated data from Appendix A. Select the applicable messages from the list displayed, and input required data in place of displayed default values. The input data will be verified and translated to a machine readable format. Range or value errors in the message data will be immediately identified to the user, so that an error may be corrected before the test is run.

e. Define the output message data requirements, from a list of segment output messages.

f. Specify the starting cycle, stopping cycle, starting frame in the cycle, number of sends in the cycle, and the number of sends in the frame as appropriate for the particular type of message (send on change or synchronous) in which the input data must be sent.

As the test data file is being constructed, it should be possible to display the current layout of the test data file, in order to modify an existing message. The resulting message data display should be identical to the display shown during the creation of a message, except that the data values will be as stored instead of the default data values. The user should also have the capability to change the message data object values back to their default values during the editing process. The default data values for each message in the system are defined by the segment builders/designers.

It should also be possible for the user to construct expected results data files for comparison to the actual test results to enhance the data analysis effort. This file should be constructed similar to the input test data files, except that the messages and data shown will be the outputs of the segment under test.

5.4.1.2 Segment Test Control. The segment test tool coordinates all communication with the segment under test and stores the segment's responses in an output data file. Additionally, the segment test tool generates the segment synchronization message. The test tool software connects to the segment's VNET and communicates with the segment through the Application Services.

5.4.1.3 Output Test Data Formatting. The results of the test should be assembled into a user friendly (human readable format) report based on the user specified interface messages. The format for this data should match the input test data and the expected results data.

5.4.1.4 Application Tests. Application tests should be designed to verify interface and segment specification requirements compliance. They should verify proper segment outputs for a given set of inputs. These tests should also determine a segment's ability to process bad data, such as input values out of range, and improper state transitions. For completeness, the segment test input data should stimulate the segment as if it were installed in the target simulator. In effect, the segment test inputs should match those expected from the simulation system during normal operation.

5.4.2 Network Analyzer. It is recommended that every modular simulator have the capability to collect data directly from the VNET. This capability is essential to analyze system problems during hardware/software integration. In order to collect data from the VNET, a network analyzer should be used. The network analyzer is used to support several of the tests described in Section 5.4.3.

5.4.3 Other tests. Based on Mod Sim experience, engineering tests must be conducted to verify the VNET interface with each segment, and the spare capacities of each segment.

5.4.3.1 VNET Interface Test. VNET interface tests must be conducted to verify that the VNET interface and Application Services are functioning properly.

5.4.3.1.1 Message Count. It is important for each of the segments to keep a running count of each type of message sent and received. These counts may be used to confirm that a given message was received as many times as it was sent, which is useful during integration. A segment's erroneous operation can often be traced to a message not being sent or the message not being received. This tool can also be used to identify a bad network interface. This feature may be implemented in Application Services. If a message arrives at Application Services, it can be assumed to be available to the Application. It is recommended that this message count be monitored until integration is complete.

A second message count may be kept by the Application Executive, when it calls Put and Get, and when compared with the count kept by Application Services, may be used to debug the Application Services. The two counts should always match if Application Services is functioning properly. Once Application Services is working, this second count can be eliminated from the code.

5.4.3.1.2 Message Capture. It is often useful to capture an individual message at the Application Services level to verify that the message was received exactly as it was transmitted or to allow analysis of segment interaction problems. This capability may be used to verify that a simulation function operates properly, when there is no other means of verification. For example, a Mod Sim program requirement existed to drop a gravity bomb, but no means existed of demonstrating the weapon release, since the weapon

displays were not available. By capturing and displaying several seconds of bomb messages, it was possible to demonstrate that the simulator met the requirement.

5.4.3.1.3 Message Latency. On the Mod Sim program, a test was conducted to measure message latency on the VNET. In response to a clock interrupt, an application segment sent a single message at a fixed rate. Likewise, the receiving segment requested that the VNET Interface interrupt the Application when that message arrived. Therefore, the sending segment was interrupted when the message was sent, and the receiving segment was interrupted when the message was received. The time difference, measured by connecting an oscilloscope to the appropriate interrupt lines on the segments, constituted the latency of the message.

5.4.3.2 Execution Timing. Simulation software is typically required to adhere to specified spare timing, spare size, spare input/output, and spare storage capacities. In order to verify these parameters, it is recommended that software test tools be implemented.

The capability to measure frame execution times should be selectable, so that frame times may be measured at critical points in the simulation, such as during simulated flight within ground effects. The timing tool should also provide subframe timing data, that may be used to identify code that takes too long to run. The data should be displayed as an array of elapsed frame times. On the Mod Sim program, the Start_Timing and Stop_Timing utilities were used to measure software execution times. These tools displayed the maximum, minimum, average and a histogram of the timing points.

5.4.3.3 Data Analysis and Display. The tools discussed in Section 5.4.3.1 and 5.4.3.2 must have the capability to display the data. The data capture tools operate in real-time, and may store the data in a known address. The data display tools typically operate as background tasks. A graphical display is often the most informative way to display data. Many data-graphing tools are available. The Mod Sim Demonstrator program used the PV~Wave commercial data-graphing tool on the Sun computer.

It is desirable to have the outputs of the measurement tools be exactly the format expected by the data-graphing tool, but this is not always possible. For example, it is useful to plot the outputs of the Segment Tester, but the Segment Tester's outputs, which consist of entire messages received from the segment, contain much more data than is commonly required. On the Mod Sim program, it was necessary to filter the data to include only the points of interest. This filtering used the Unix utility "sed" (stream editor) and some special-purpose routines written in C. The "awk" and "perl" utilities would also be good choices for these filter routines.

6. SEGMENT SPECIFIC CONSIDERATIONS

Design requirements for any segment include the simulator application functions and the segment support functions. Obviously the simulator application function requirements are unique for each segment. Conversely the support functions are generally common across all segments and should probably be specified at the system level if for no other reason than to promote reusability. However, system level requirements imposed on segments need to be carefully considered so as to not unnecessarily restrict or encumber the segment designer. For example, a Visual segment, which normally has limited application functionality, may not warrant a full set of common support functions and the associated complement of processing resources.

The following subparagraphs briefly describe the overall functionality of the individual segments and refer to tables that address the individual functions within the segments. The tables provide information with respect to the potential reuse of the individual functions, indications of any special equipment normally associated with the function, and any special considerations for the function. Reuse potential is expressed in terms of high, moderate and low and generally indicates whether a function implementation is adaptable to many varied applications, several related applications or an isolated set of applications respectively.

6.1 Flight Station Segment. The Flight Station segment provides for the simulation of the common aircraft systems such as the electrical system, hydraulic system, fuel management system, pneumatic system, oxygen system, autochecklist and crew station interface functions. See Table 6.1-1 for further information on the segment functions.

6.2 Flight Controls Segment. The Flight Controls segment provides for the simulation of the ownship control surfaces, control devices, and control systems. The segment models the movement of control surfaces in response to change in the settings of the flight controls. That information is fed to internal models of automatic flight control systems, stability enhancement systems and trim. See Table 6.2-1 for further information on the segment functions.

6.3 Flight Dynamics Segment. The Flight Dynamics segment provides the simulation of the flight characteristics, flight performance, flying/handling qualities, mass properties and structural limits of the aircraft throughout the flight envelope of the ownship. The flight characteristics, flight performance and flying qualities model simulates the air vehicle motion using flight test data, wind tunnel data and derived data as model inputs. See Table 6.3-1 for further information on the segment functions.

6.4 Propulsion Segment. The Propulsion segment provides for the simulation of the core engine, thrust generation, starting system, engine inlet, engine fuel, emergency/ auxiliary power unit and engine exhaust gas functions throughout the flight envelope of the ownship. See Table 6.4-1 for further information on the segment functions.

6.5 Navigation/Communication Segment. The NAVCOM segment simulates the navigation and communication system functions within the MSS. The navigation

functions simulate navigational guidance with respect to selected navigation aid station. The communications functions simulate reception and transmission of voice and encoded communications for the ownship. See Table 6.5-1 for further information on the segment functions.

6.6 Weapons Segment. The Weapons segment simulates the offensive weapon system functions employed on the ownship including weapons delivery, stores management , target designation and tracking, and weapons dynamics and effects. See Table 6.6-1 for further information on the segment functions.

6.7 Radar Segment. The Radar segment simulates the radar functions, including system operational capabilities, radar effects, image mapping, and video generation and display. See Table 6.7-1 for further information on the segment functions.

6.8 Electronic Warfare Segment. The Electronic Warfare segment simulates the threat detection, electronic counter and counter-counter measures, expendable counter measures and warning functions of the ownship defensive avionics. See Table 6.8-1 for further information on the segment functions.

6.9 Physical Cues Segment. The Physical Cues segment provides for the simulation of environmental sounds, aircraft motion, and vibration and buffet functions throughout the flight envelope of the ownship. See Table 6.9-1 for further information on the segment functions.

6.10 Visual Segment. The visual segment provides the trainee visual out-the-window references and cues relating to the tactical and natural environments through which the ownship is operating. The visual segment may also provide visual cues for profile type view equipment such as Forward Looking Infrared (FLIR) or Infrared Search and Track (IRST). See Table 6.10-1 for further information on the segment functions.

6.11 Instructor/Operator Station Segment. The Instructor/Operator Station segment provides the central point of control and monitoring for the entire simulation and related training activities. As the central point of control for the MSS, the IOS provides the capability to control the state of the simulation (freeze, reset, etc.), interject malfunctions into the simulation, change parameters within the simulation, plan missions and lessons, monitor the state of the simulation and monitor/measure trainee performance. The IOS segment has complete control of the simulator; therefore, it may perform other specific and unique functions. These functions include initiation of maintenance tasks such as daily readiness tests and diagnostics and to control and manage specific test functions such as hardware/software integration and/or formal qualification testing up through and including FAA type acceptance testing and simulator certification (SIMCERT) testing. See Table 6.11-1 for further information on the segment functions.

6.12 Environment Segment. The Environment segment performs the functions necessary to simulate the tactical and natural environments external to the ownship during autonomous operations. The Environment segment also provides the functional interface for integrating the MSS into a Multiple Simulator Environment (MSE). See Table 6.12-1 for further information on the segment functions.

Function	Reuse Potential	Special Equipment	Function Considerations
Electrical System	Low: Power generation and distribution tend to be unique to different aircraft types		
Hydraulic System	Low: Hydraulic systems tend to be unique to different aircraft types		
Fuel Management System	Low: Fuel management systems tend to be unique to different aircraft types		
Pneumatic System	Low: Pneumatic systems tend to be unique to different aircraft types		
Autochecklist	Low: Autochecklist functions tend to be unique to different aircraft types		
Oxygen System	Low: Oxygen systems tend to be unique to different aircraft types		
Crew Station Interface	Low: Crew station interfaces tend to be unique to different aircraft types	Crew station interface equipment for controls & displays	Use of actual avionics equipment for controls & displays may necessitate a backdoor interface to other segments due to avionics coupling requirements.

Table 6.1-1
Flight Station Segment Functions

Function	Reuse Potential	Special Equipment	Function Considerations
Primary Controls	Moderate to high: Function generic for aircraft with standard controls; Function lends itself to data-driven implementation	Control loading equipment, if active loading is required; Interface to avionics flight control computer, if required	Time critical function; Must consider latency and transport delay in conjunction with Flight Dynamics segment; Use of actual avionics flight control processors may impose stringent latency requirements. If aircraft includes thrust vectoring, must consider latency in conjunction with propulsion segment.
Secondary Control Devices	Moderate to high: Function generic for aircraft with standard controls; Function lends itself to data-driven implementation	Control loading equipment, if active loading is required	
Trim	Moderate: Function generic for aircraft with standard controls		
Hinge Moments	High: Function generic across multiple applications; Function lends itself to data-driven implementation		
Automatic Flight Controls System	Low: AFCSs tends to be unique to different aircraft types	Interface to avionics flight control computer, if required	Time critical function; Must consider latency and transport delay in conjunction with Flight Dynamics segment; Use of actual avionics flight control processors may impose stringent latency requirements.
Toe Brakes and Anti-Skid	High: Function generic across multiple applications	Control loading equipment, if active loading is required	

Table 6.2-1
Flight Controls Segment Functions

Function	Reuse Potential	Special Equipment	Function Considerations
Forces and Moments	Moderate: Although coefficient buildup can be standardized, the actual tables and effects will always be peculiar to the simulated aircraft	High fidelity aerodynamics models may necessitate array processing or high-speed floating point processing equipment	Time critical function; Must consider latency and transport delay in conjunction with Flight Controls segment
Equations of Motion	High: Function tends to be generic across similar aircraft types		Earth axis coordinate system reference must be reconciled across other related segments (Visual, Radar, EW, NavComm, etc) that perform spatial calculations. Embedded avionics earth axis reference must be considered.
Weight and Balance	High: Function tends to be generic; Function lends itself to data-driven implementation		
Envelope Violation	High: Function tends to be generic across similar aircraft types; Function lends itself to data-driven implementation		

Table 6.3-1
Flight Dynamics Segment Functions

Function	Reuse Potential	Special Equipment	Function Considerations
Core Engine System	Low: Function is unique to engine type and performance between common types	Interface to aircraft engine controller, if required	
Thrust Generation System	Moderate to low: Function somewhat generic across similar propulsive systems	Interface to aircraft engine controller, if required	
Starting System	Low: Too many different start methods to consider standardization of the function		
Engine Inlet System	Moderate to high: Function tends to generic to multiple applications		
Engine Fuel System	High: Function generic to multiple applications		
Engine Bleed Air System	Moderate to high: Function tends to be generic to multiple applications		
Transmission System	Moderate: Function tends to be generic across many engine types		
Engine Oil System	High: Function is generic to multiple applications		
Emergency/Auxiliary Power Unit	Moderate: Function somewhat generic across power unit systems		
Engine Exhaust System	Low: Function typically unique to exhaust system type		If aircraft includes thrust vectoring, must consider latency in conjunction with Flight Controls segment

Table 6.4-1
Propulsion Segment Functions

Function	Reuse Potential	Special Equipment	Function Considerations
Attitude Heading Reference System	Moderate: Function tends to be generic across applications		
Inertial Navigation System	High: Function is generic across multiple applications		
Radar Altimeter	High: Function is generic across multiple applications		
Radio Navigation Aid Systems	High: Function is generic across multiple applications	Nav Aid audio signal generation and distribution equipment	Advanced audio systems can provide environmental and communication audio processing. Equipment capabilities should be considered when allocating function to this segment and to the Physical Cues and EW segments.
Communica[illegible]	Moderate: Function is somewhat generic across applications	Voice communication amplification, processing & distribution equipment	Advanced audio systems can provide environmental and communication audio processing. Equipment capabilities should be considered when allocating function to this segment and to the Physical Cues and EW segments.
Star Tracker	High: Function is generic across multiple applications		
Doppler Radar	High: Function is generic across multiple applications		
Air Data System	High: Function is generic across multiple applications		
Application Unique Avionics Simulation	Low: Function is unique to specific avionics system	Interface to aircraft avionics computers, if required	
Identification Friend or Foe	High Function is generic across multiple applications		

Table 6.5-1
Navigation/Communication Segment Functions

Function	Reuse Potential	Special Equipment	Function Considerations
Ownship Fire Control	Low: Highly dependent on fire control system performance and weapons complement	Interface to actual avionics for fire control processing, if required	
Ownship Weapon Dynamics	Moderate: Individual weapon models should be common across aircraft that employ the weapon		
Ownship Stores	Low: Dependent on aircraft weapons carriage capabilities and configurations		
Target Designation	High: Function is generic across multiple applications		
Threat Weapon Damage Assessment	High: Function is generic across multiple applications		

Table 6.6-1
Weapon Segment Functions

Function	Reuse Potential	Special Equipment	Function Considerations
Radar Processor	Low: Performance and mode control is unique between radar types and is dependent on aircraft avionics complement	Interface to actual avionics for radar control and/or processing, if required	
Radar Image Generation	High: Modelling of radar propagation & effects is generic for given radar modes across multiple radar systems; Function lends itself to data-driven implementation.	Radar image generation equipment	
Airborne Interrogate Sensor	High: Function is generic across multiple applications		
Radar Database Management	High: Function is coupled to radar system image generation equipment	Radar database storage media equipment	This is a service function. It may be allocated to this segment or to the Visual or Environment segments based on application-specific trade study factors.
Radar Guidance	Low: Guidance imaging and performance characteristics for TF & TA tend to be aircraft specific	Interface to actual avionics for radar guidance processing, if required	
Mission Computer	Low: Functionality tends to be unique across aircraft types	Interface to actual avionics for mission processing, if required	
Aircraft System Interface	Low: Functionality tends to be unique across aircraft types		
Crew Station Hardware Interface	Low: Dependent on crew station display configuration	Crew station hardware interface equipment	
Spatial Relations	High: Function is generic across multiple applications	Array processing or high-speed floating point processing equipment	This is a service function. It may be allocated to this segment or to the Visual or Environment segments based on application-specific trade study factors.
Occulting	High: Function is generic across multiple applications	Array processing or high-speed floating point processing equipment	This is a service function. It may be allocated to this segment or to the Environment segment based on application-specific trade study factors.

Table 6.7-1
Radar Segment Functions

Function	Reuse Potential	Special Equipment	Function Considerations
Expendable Countermeasures (EXCM)	Moderate: Function is somewhat generic across multiple applications	Interface to actual avionics for EXCM processing, if required	
Dedicated Displays	Low: Dependent on unique display requirements	EW display generation equipment	
Electronic Countermeasures (ECM)	Moderate: Reusable across aircraft that use a particular EW system	Interface to actual avionics for ECM processing, if required	
Warning Receiver	Moderate: Reusable across aircraft that use a particular EW display	Interface to actual avionics for RWR processing, if required	
Threat Detection	Moderate: Reusable across aircraft that use a particular EW system	Interface to actual avionics for threat detection processing, if required	

Table 6.8-1
Electronic Warfare Segment Functions

Function	Reuse Potential	Special Equipment	Function Considerations
Environmental Sound	High: Function generic across many applications	Audio signal generation equipment	Advanced audio systems can provide environmental and communication audio processing. Equipment capabilities should be considered when allocating function to this segment and to the Nav/Comm and EW segments.
Anti G-suit	High: Function generic across many applications	Anti G-Suit & interface equipment	
G-seat	High: Function generic across many applications	G-Seat & interface equipment	
Motion	High: Motion cueing is generic for air vehicles, but requires tuning for varying performance.	Motion platform & interface equipment	
Vibration and Buffet	High: Function generic across many applications	Seat shaker or motion system with high frequency response capability	

Table 6.9-1
Physical Cues Segment Functions

Function	Reuse Potential	Special Equipment	Function Considerations
Image Generation	High: Function is generic across multiple applications.	Image generation equipment	
Moving Models	High: Function is generic across multiple applications.	Typically implemented by image generation equipment	
Visual Database	High: Function is coupled to image generation	Visual database storage media equipment	This is a service function. It may be allocated to this segment or to the Environment or Radar segments based on application-specific trade study factors.
Visual Scene Environment	High: Function is generic across multiple applications.	Typically implemented by image generation equipment	
Lighting Control	High: Function is generic across multiple applications.	Typically implemented by image generation equipment	
Spatial Relations	High: Function generic across multiple applications	Array processing or high-speed floating point processing equipment; Can be implemented by image processor.	This is a service function. It may be allocated to this segment or to the Environment or Radar segments based on application-specific trade study factors.
Occulting	High: Function generic across multiple applications	Array processing or high-speed floating point processing equipment; Can be implemented by image processor.	This is a service function. It may be allocated to this segment or to the Environment or Radar segments based on application-specific trade study factors.
Mission Computer/Display Processor Interface	Low: Functionality tends to be unique across aircraft types	Interface to actual avionics for mission processing, if required	
Visual Crew Station Interfacing	Moderate: Dependent on crew station display configuration	Visual crew station interface equipment	
Visual Display Systems	Varies from Low to High: Dependent on specific display technology and field of view training requirements	Display & interface equipment	

Table 6.10-1
Visual Segment Functions

Function	Reuse Potential	Special Equipment	Function Considerations
Ownship Status and Control	High: Standard MSS data definition for IOS parameters	User interface equipment for control, monitoring & display	Generic IOS must accommodate variabilities such as user presentation format & aircraft systems configuration.
Ownship Malfunction	High: Standard MSS data definition for IOS parameters	User interface equipment for control, monitoring & display	Generic IOS must accommodate variabilities such as user presentation format & aircraft failure modes.
Ownship Controls Disagreement	High: Standard MSS data definition for IOS parameters	User interface equipment for control, monitoring & display	Generic IOS must accommodate variabilities such as user presentation format, aircraft control configuration.
Navigation/Communication Status and Control	High: Standard MSS data definition for IOS parameters	User interface equipment for control, monitoring & display	Generic IOS must accommodate variabilities such as user presentation format, aircraft configuration & simulator comms.
Environment Status and Control	High: Standard MSS data definition for IOS parameters	User interface equipment for control, monitoring & display	Generic IOS must accommodate variabilities such as user presentation format & environment model capabilities
Mission Status and Control	High: Standard MSS data definition for IOS parameters	User interface equipment for control, monitoring & display	Generic IOS must accommodate variabilities such as user presentation format & training progression.
Simulator Control	High: Standard MSS data definition for IOS parameters	User interface equipment for control, monitoring & display	Generic IOS must accommodate variabilities such as user presentation format & simulator control capabilities.
Crew Performance Monitoring and Measurement	High: Standard MSS data definition for IOS parameters	User interface equipment for control, monitoring & display	Generic IOS must accommodate variabilities such as user presentation format & training metrics & indicators.

Table 6.11-1
Instructor/Operator Segment Functions

Function	Reuse Potential	Special Equipment	Function Considerations
Atmosphere	High: Function is generic across multiple applications		
Database Management	Low: Function is dependent on target Visual, Radar and Sensor simulation database architectures		
Navigation Database	High: Function is generic across multiple applications	Navigation database storage media equipment	
Spatial Relations	High: Function is generic across multiple applications	Array processing or high-speed floating point processing equipment	This is a service function. It may be allocated to this segment or to the Visual or Radar segments based on application-specific trade study factors.
Visual Database	Moderate: Implementation is highly dependent on visual system image generation equipment	Visual database storage media equipment	This is a service function. It may be allocated to this segment or to the Visual segment based on application-specific trade study factors.
Radar Database	Moderate: Implementation is highly dependent on radar system image generation equipment	Radar database storage media equipment	This is a service function. It may be allocated to this segment or to the Radar segment based on application-specific trade study factors.
Occulting	High: Function is generic across multiple applications	Array processing or high-speed floating point processing equipment	This is a service function. It may be allocated to this segment or to the Visual or Radar segment based on application-specific trade study factors.
Ownship Weapons Damage Assessment	Moderate: Damage assessment for a given weapon should be generic, but the ownship complement of weapons carriage is typically unique		
Entity Database	High: Function is generic across multiple applications		
Entity Management	High: Function is generic across multiple applications		

Table 6.12-1
Environment Segment Functions

Entity Weapons	High: Function is generic across multiple applications		
Entity Expendable Countermeasures	High: Function is generic across multiple applications		
Multiple Simulator Environment (MSE) Interaction	High: Function is generic across applications for a given MSE network protocol (e.g. SIMNET, DIS, BDS-D, etc.)	MSE network interface equipment	

Table 6.12-1
Environment Segment Functions

7. SUPPORTING INFORMATION

7.1 Networking Standards. There are several industry standards which address the network protocols and interfaces. These include the International Organization of Standards Open Systems Interconnection (ISO/OSI) model, the ANSI FDDI standard, IEEE standards, and Distributed Interactive Simulation (DIS).

7.1.1 ISO/OSI Model. A number of different standards organizations have divided the operations of networked communications into layers or partitions. One noted example is the OSI interconnection model defined in ISO-7498. This model consists of seven protocol layers as depicted in Figure 7.1.1-1. While the exact implementation is not universally accepted, the functionality and protocols defined therein are.

A fundamental premise in ISO-7498 is the notion of peer entities at each layer communicating with one another. That is two Application Layer entities communicating with one another, two Physical Layer entities communicating with each other, and so on. Each level of entity has its own requirements and its own expectations of higher and lower entities. For example, Application Layer entities may safely assume that their lower level Presentation Layer entities will supply them with data that is in a form that is understandable to them.

Whether or not the layering is strictly followed, as with a system that has a separate software element for each layer's entity, or whether it is followed more informally as it is in the Internet model and in the Mod Sim architecture, the ISO/OSI model allows the requirements for communications between processes to be more easily organized and understood. The following paragraphs describe functionality of the individual ISO layers.

7.1.1.1 ISO Application Layer. This is the only Layer in the ISO/OSI model that provides services directly to Applications. No Application may take advantage of a service provided in another Layer, except through an Application service. The services in the Application Layer include:

a. Transfer of information.

b. Identification of intended communication partners.

c. Establishment of authority to communicate.

d. Identification of constraints on data syntax.

The last point means that it is the responsibility of each Application Layer to make its own data formatting requirements known to the Presentation Layer so that the Presentation Layer can provide them.

In the Mod Sim architecture, the services provided in the Application Layer are encapsulated in a package called Application Services. Individual segment Applications may only communicate with one another through these Application Services.

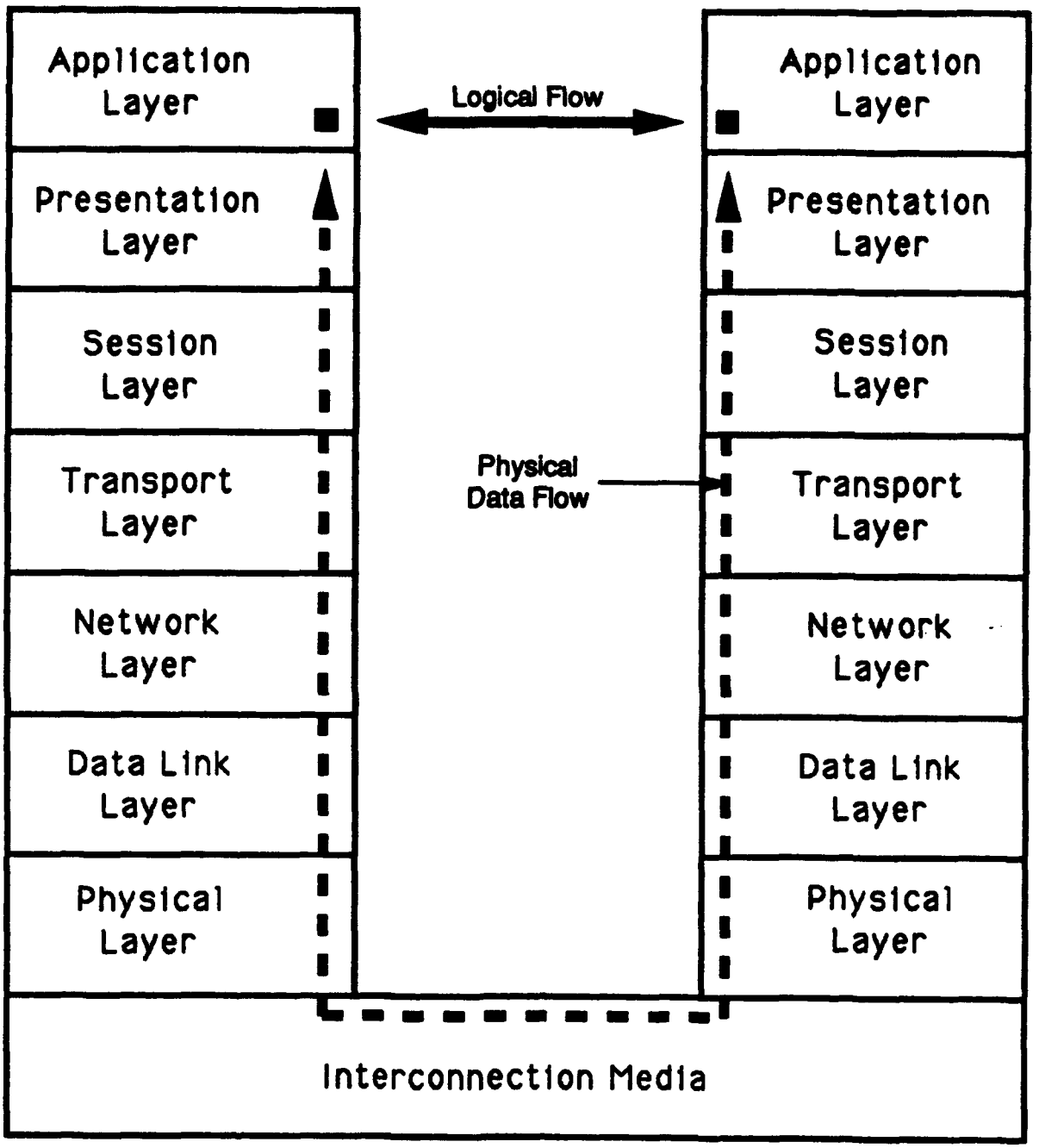


Figure 7.1.1-1
ISO/OSI Reference Model

Since message data in the Application Layer is purely data, discussion of headers is meaningless. Applications may wish to have identifiers within the message that control how the message is processed, but that is purely a matter for the designer of the Application, and is beyond the scope of this discussion. Figure 7.1.1.1-1 shows the headers for a typical TCP/IP message transmitted over FDDI.

It is difficult to separate functions into layers in the real world. For example, in the FDDI board driver for the Interphase 4211 board, the sending host is required to build the MAC header and to allocate space for the MAC trailer before it sends the packet to the board; but the MAC trailer is stripped off by the hardware on the receiving station and is only available by reading certain status registers on the board. So, does the processing of the MAC header and trailer reside on the FDDI board or in the host driver software? It depends on whether the sender or receiver of a message is being discussed. Similar caution should apply to all the statements about functions and layers above.

7.1.1.2 ISO Presentation Layer. The Presentation Layer provides for the representation of information that Applications either communicate or refer to in their communications. For example, some CPUs are Big-Endian and others are Little-Endian (see Section 4.3.1.2). It is the responsibility of the Presentation Layer to do any conversion required between the layers.

Above the Presentation Layer, each Application deals with data that makes logical sense to it. If a Big-Endian machine and a Little-Endian machine each add two integers or two floating-point numbers, then display the numbers and results to one another through a network connection, each machine should agree that the computation is correct.

Below the Presentation Layer a common method of data representation is used. In the case of a Big-Endian machine and a Little-Endian machine, it is likely that only one of them would understand this data. While the ISO/OSI model calls only for "negotiation" between the Presentation Layers on two hosts to determine the lower-level representation to be used, the Mod Sim architecture is defined such that Big-Endian integers and addresses and IEEE-754 floating point numbers will be used exclusively below the Presentation Layer.

Headers are not commonly applied to the message by the Presentation Layer.

7.1.1.3 ISO Session Layer. The Session Layer provides a "session connection" between two presentation entities and permits normal data exchange. This data exchange can either be connection-based or connectionless, depending on the kind of data transfer. For example, Universal Datagram Protocol (UDP) is connectionless and is the only protocol of the three in which broadcasting is commonly done. Transmission Control Protocol (TCP) and Xpress Transfer Protocol (XTP) are connection-based and support both point-to-point and multicast (one point to many) data transfer.

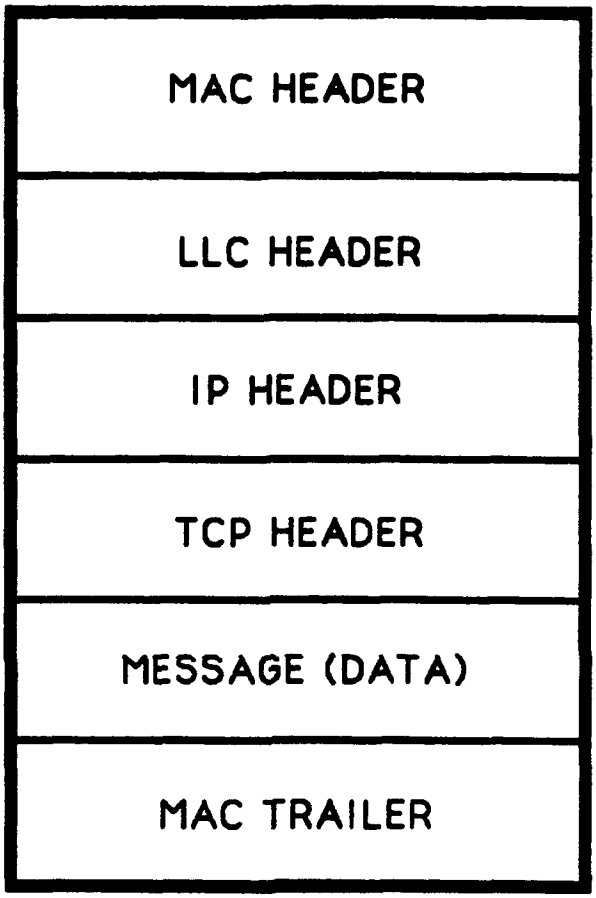


Figure 7.1.1.1-1
TCP/IP Message Transmitted Over FDDI

While UDP, TCP and XTP all function at the Transport and Network Layers, it is the responsibility of the Session Layer to set up the connections that enable their operation. The software in the Mod Sim architecture to do this resides in the VNET interface.

The Mod Sim program did not apply any particular header at the Session Layer, but rather relied on the port number in the Transfer Layer header to uniquely identify the message. Future implementations of Mod Sim may find it helpful to place the Message ID and size (for variable-length messages) in a header.

7.1.1.4 ISO Transport Layer. The purpose of the Transport Layer is to provide connections between Session Layer entities and to relieve them of any concern with the detailed way in which reliable and cost-effective transfer of data is achieved. A connection at the Transport Layer may support several different Session Layer connections, or a Session Layer connection may be implemented by several consecutive Transport Layer connections. Several protocols perform most of their work at the Transport Layer. The protocols of interest are again TCP, UDP and XTP.

UDP is connectionless: no acknowledgment is required when a message is transmitted from one process to another. The sending process merely transmits the message to the network. It is entirely possible that the receiving process may not be able to receive the data, or that the data may contain errors. No check is made at the Transport layer for either of these eventualities, though the Session Layer or Application Layer may wish to perform checks (e.g., the Application may require that its corresponding Application acknowledge all messages).

TCP is connection-based: before one process transmits data to another, the two communicate with one another to establish that data transmission is possible. After the data is transmitted, the receiving process acknowledges its receipt with another message. The closing down of a connection is accomplished with a similar exchange of messages. It is the responsibility of the Session Layer to set up connections, manage data transfer, allow for the orderly shutdown of connections, and synchronize the communications (determining which process transmits at what time).

XTP is similarly connection-based, but the number and size of acknowledgments are smaller than those in TCP. XTP can at least theoretically exchange data faster than TCP.

TCP and UDP both require Internet Protocol (IP) to meet their Network Layer requirements. For example, the TCP or UDP header of a message which is placed on the message by the Transport Layer will be prefaced with an IP header by the Network Layer. XTP performs its own Network Layer services. An XTP packet contains both a header and a trailer and does not require any IP header.

7.1.1.5 ISO Network Layer. The Network Layer provides an addressing mechanism for the transfer of data. The most commonly used addressing method is the 32-bit, IP address. MIL-STD-1777 describes this addressing mechanism and illustrates the other functions performed by the Internet Protocol.

The Network Layer (assuming IP is used) places an IP header inside the MAC and Logical Link Control (LLC) headers on the message which contains the IP addresses of the sender and intended receiver(s) and fields containing a flag for the protocol contained in the packet, packet length, and packet checksum

The bulk of the function of the Network Layer has to do with routing through different networks; this is not done in the Mod Sim architecture.

7.1.1.6 ISO Data Link Layer. Data link connection takes place by means of physical connection. It is always true that higher level layers depend on lower level layers for their functionality. Data Link Layer functions include token management on a ring bus architecture, system timer support, packet framing, and peer-to-peer communications using hardware addresses. These functions correspond to the FDDI MAC layer. Data Link Layer data structures correspond to FDDI or IEEE-802 frames.

The MAC sub layer also provides a network-specific header (and possibly trailer) on the data which forms the outermost boundaries of a packet. This wrapper provides information which controls such things as frame type identification, frame length identification, 48-bit hardware (or MAC) addresses of sender and intended receiver, and flags which may note that an error has been detected in the system. The type of header supplied will depend on the type of network (Ethernet, FDDI, etc.) being used to transmit the data. Figure 7.1.1.1-1 shows all the headers in a typical FDDI message.

The Data Link Layer also includes the following activities: connection management, fault detection, fault isolation, and ring reconfiguration. At the top of the Data Link Layer, providing services to the Network Layer, is IEEE-802.2 Logical Link Control. The LLC sub layer supplies a header (contained inside the MAC header) containing the type of packet which is to follow and the protocol which it contains.

7.1.1.7 ISO Physical Layer. The Physical Layer contains the mechanical and electrical elements that comprise what is called the physical media. The process of establishing a physical connection between two processes involves such things as connecting cables and boards, triggering electromechanical bypasses, and providing electrical power. The unit of data on the Physical Layer is one bit (in serial transmission) which is either on or off. These functions correspond to the FDDI Physical Media Dependent (PMD) layer.

The Physical Layer also includes such things as 4B/5B encoding (encoding four bits into five line state transitions) which correspond to FDDI Physical (PHY) layer functions.

7.1.2 The Fiber Distributed Data Interface (FDDI), FDDI also has layers: the MAC Layer, the PHY Layer, the PMD Layer, and a "layer" for Station Management (SMT) which performs functions at all the other three levels. These layers perform functions that correspond to the two lowest layers of the ISO/OSI model: the MAC layer corresponds to the lower part of the ISO/OSI Data Link Layer and the PMD and PHY correspond to lower and upper portions, respectively, of the ISO Physical Layer. Figure 7.1.2-1 illustrates these layers with respect to the ISO/OSI model.

7.1.3 Institute of Electrical and Electronics Engineers (IEEE). The IEEE has also published a set of standards describing the lower two ISO/OSI layers. IEEE-802.2

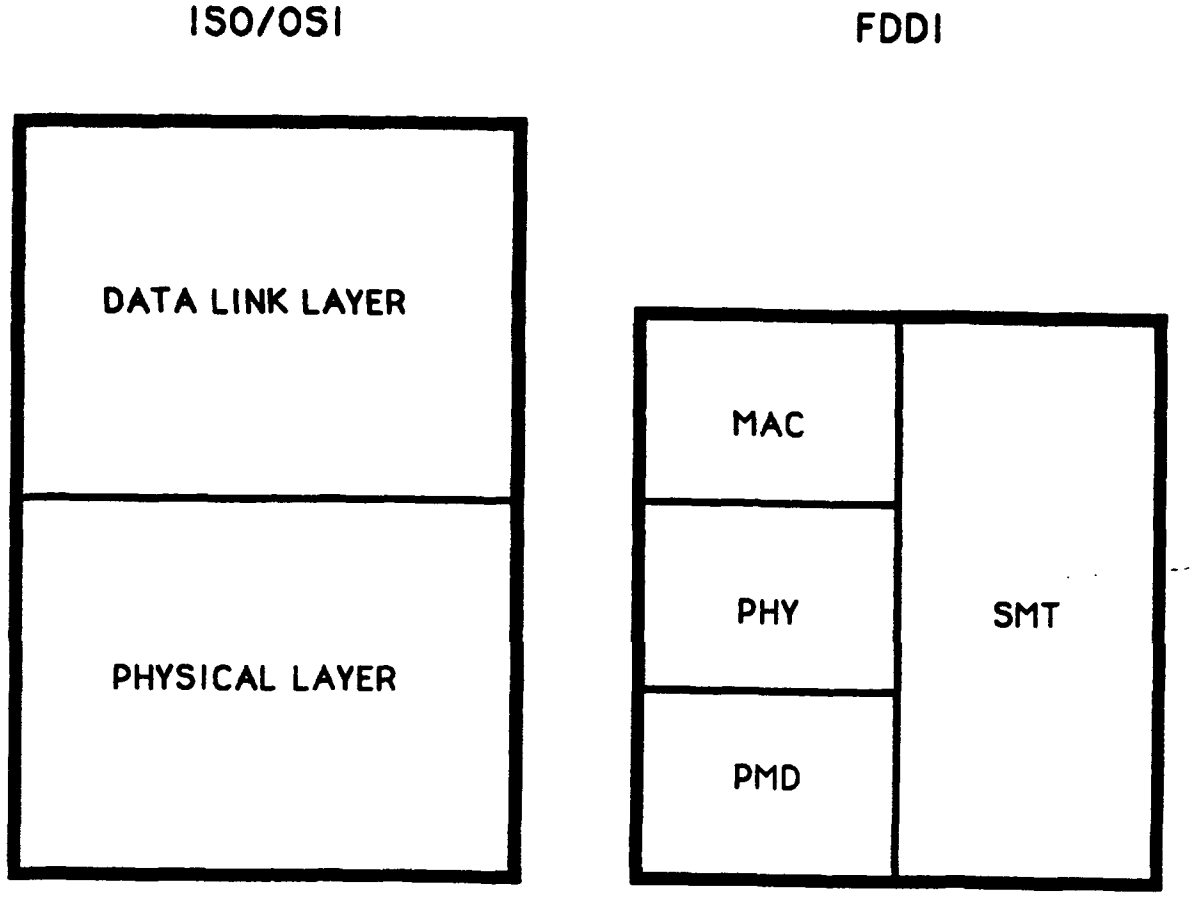


Figure 7.1.2-1
FDDI Sublayers

describes LLC which is at the "top" of the Data Link Layer; IEEE-802.2 LLC is also used above the other layers of the FDDI standard. IEEE-802.3 describes Ethernet, IEEE-802.4 describes the Token Bus, and IEEE-802.5 describes the Token Ring. Figure 7.1.3-1 illustrates the IEEE standards as they relate to the ISO/OSI model.

7.1.4 Distributed Interactive Simulation (DIS). DIS is an emerging standard for networked simulation which is intended to replace SIMNET. The draft Standard, temporarily numbered IST-CR-93-01, contains the requirements for the protocol data units (PDUs) associated with entity information, entity interactions and simulation management information that are exchanged between simulation applications interacting in a distributed interactive simulation. The DIS protocol encompasses a portion of the application layer of ISO/OSI model.

7.2 Processes. There are several software development processes which are gaining acceptance in the real-time simulation industry. Three of the more prominent are Synthesis, ADARTS and Structural Model which are discussed in the following paragraphs.

7.2.1 The Software Productivity Consortium (SPC) Synthesis Process. Synthesis is a methodology for constructing software systems as instances of a family of systems that have similar descriptions. Synthesis enables developers to exploit similarities to eliminate redundant work, and allow focusing on resolving the project unique variations. The Synthesis Guidebook, SPC-91122-MC, provides an introduction to the practice of the Synthesis methodology of software development, and a thorough discussion of Domain Engineering (standardization of Application Engineering products and processes).

7.2.2 Ada-based Design Approach for Real-Time Systems (ADARTS). For simulator programs where Ada is imposed, the ADARTS Guidebook, SPC-91104-MC, is available to assist systems designers and software engineers in the development of the simulator. The guidebook provides decomposition guidelines and heuristics, guidance in allocating the partitions resulting from the decomposition, and guidance in defining Ada support tasks, Ada task interfaces, and the packaging of concurrent processes.

7.2.3 Structural Model. A structural model is a method for specifying and implementing software system functionality. It involves partitioning of problems, coordination between components, evaluation of the structural model and the system to be built, and application of the structural model. Partitioning is the decomposition of large problems into smaller sub problems. The sum of the solutions to the sub problems represent the overall problem solution. Coordination consists of communication (sharing of computations between structural elements) and activation (cooperation between structural elements). Evaluation is performed on the structural model to determine how well it represents the system to be built, and on the system to determine if it will satisfy the required functionality and intended qualities. Application involves the explicit and rigorously enforced use of the structural model to achieve consistency throughout the system development.

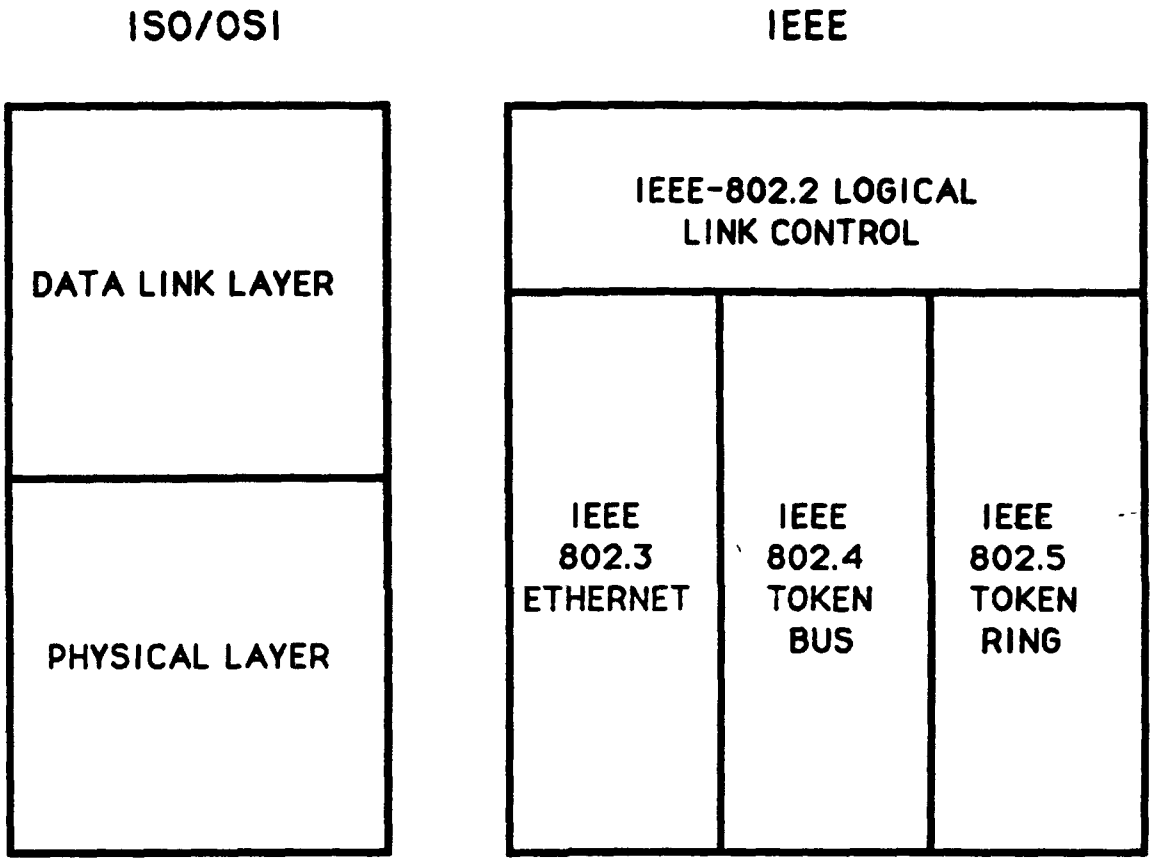


Figure 7.1.3-1
IEEE Standards

At the 1992 Industry/Interservice Training Systems and Education Conference (I/ITSEC), the Software Engineering Institute of Carnegie Mellon University presented a paper, sponsored by the Department of Defense, on structural models for real-time simulation. The paper is a precursor to an engineering guidebook to be commissioned in 1993, which will provide details on successful structural models and associated development methods. In the future, the Air Force will expect bidders to describe the proposed structural model(s) to be used on a project, demonstrate that the model(s) is complete with respect to required functionality and intended qualities, and how the model(s) will be applied consistently across the project.

8. ACRONYMS

ACRONYM	DEFINITION
ADARTS	Ada-based Design Approach for Real-Time Systems
AFB	Air Force Base
AMD	Advanced Micro Devices
AMIC	Automated Module Interface Compiler
BIU	Bus Interface Unit
CDR	Critical Design Review
CPU	Central Processing Unit
CSC	Computer Software Component
CSCI	Computer Software Configuration Item
CSU	Computer Software Unit
DIS	Distributed Interactive Simulation
DSSA	Domain Specific Segment Architecture
FAA	Federal Aviation Administration
FDDI	Fiber Distributed Data Interface
GF	Growth Factor
GUI	Graphical User Interface
HSI	Hardware Software Integration
IDD	Interface Design Document
IEEE	Institute of Electrical and Electronics Engineers
IOS	Instructor/Operator Station
IP	Internet Protocol
IRS	Interface Requirements Specification
ISO/OSI	International Organization for Standards/Open Systems Interconnection
ISWG	Interface Standard Working Group
LLC	Logical Link Control
MAC	Media Access Control
MFD	Multifunction Display
ML	Maximum Length
Mod Sim	Modular Simulator
MSS	Modular Simulator System
MTU	Maximum Transfer Unit
OS	Operating System

PDR	Preliminary Design Review
PDU	Protocol Data Unit
PHY	Physical layer
PL	Protocol Header Length
PMD	Physical Media Dependent layer
RAM	Random Access Memory
SIMNET	Simulation Network
SMT	Station Management
SPC	Software Productivity Consortium
SSS	System/Segment Specification
S&TS	Simulation and Training Systems
TCP	Transmission Control Protocol
TO	Technical Order
UDP	Universal Datagram Protocol
USAF	United States Air Force
UTSS	Universal Threat Simulation System
VME	Versa Module European
VNET	Virtual Network
WST	Weapon System Trainer
XTP	Xpress Transfer Protocol

ACTIVE PAGE RECORD

PAGE NO.	REV LTR	ADDED PAGES				PAGE NO.	REV LTR	ADDED PAGES			
		PAGE NO.	REV LTR	PAGE NO.	REV LTR			PAGE NO.	REV LTR	PAGE NO.	REV LTR
1	-	40	-	79	-						
2	-	41	-	80	-						
3	-	42	-	81	-						
4	-	43	-	82	-						
5	-	44	-	83	-						
6	-	45	-								
7	-	46	-								
8	-	47	-								
9	-	48	-								
10	-	49	-								
11	-	50	-								
12	-	51	-								
13	-	52	-								
14	-	53	-								
15	-	54	-								
16	-	55	-								
17	-	56	-								
18	-	57	-								
19	-	58	-								
20	-	59	-								
21	-	60	-								
22	-	61	-								
23	-	62	-								
24	-	63	-								
25	-	64	-								
26	-	65	-								
27	-	66	-								
28	-	67	-								
29	-	68	-								
30	-	69	-								
31	-	70	-								
32	-	71	-								
33	-	72	-								
34	-	73	-								
35	-	74	-								
36	-	75	-								
37	-	76	-								
38	-	77	-								
39	-	78	-								

REVISIONS			
LTR	DESCRIPTION	DATE	APPROVAL